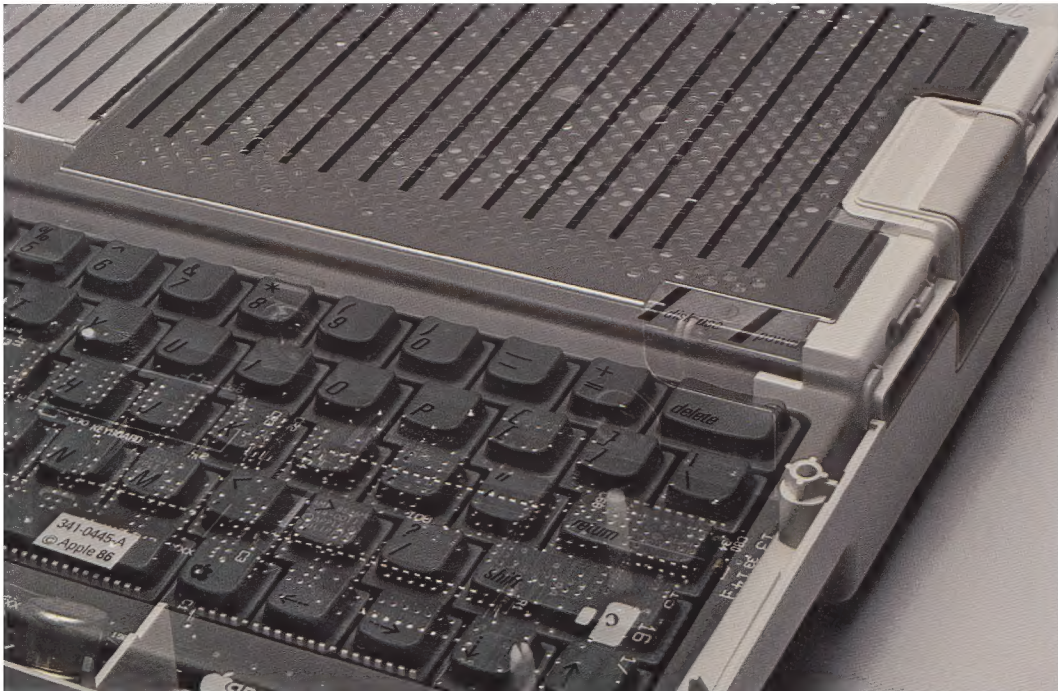




Apple<sup>®</sup> II

# Apple IIc Technical Reference Manual



Includes ROM Listings for Memory Expandable IIc

> \$24.95 FPT  
USA

## Apple® Technical Library Titles for the Apple IIe and IIc

### The Official Publications from Apple Computer, Inc.

Apple IIe and Apple IIc programmers, developers, and enthusiasts will find a wealth of information in the Apple Technical Library, an ongoing series of comprehensive reference manuals. The first volumes in the Library contained detailed information about the Apple IIe and Apple IIc computers. They describe the hardware, firmware, the ProDOS 8 operating system, and the Applesoft BASIC programming language found in Apple IIe and IIc computers.

These books, written and produced by Apple Computer, Inc., provide definitive references for those interested in getting the most out of their Apple IIe or IIc.

Apple Technical Library Titles for the Apple IIe and IIc include:

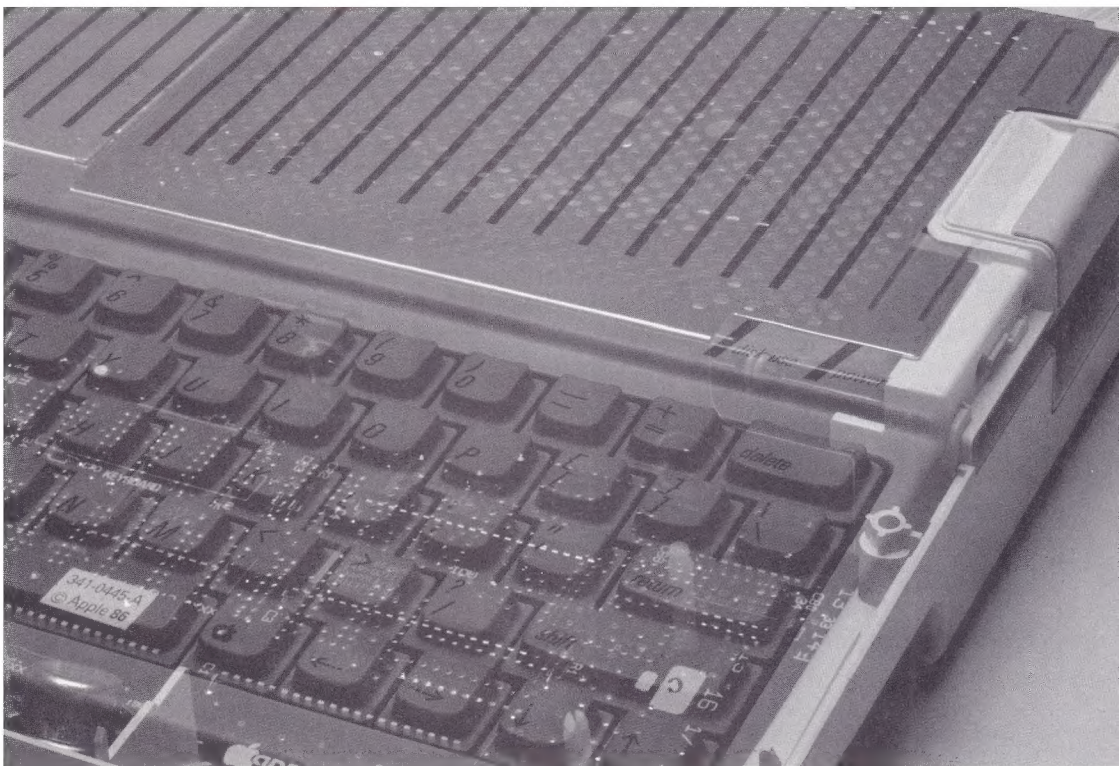
- Apple IIe Technical Reference
- Apple IIc Technical Reference
- Applesoft Tutorial
- Applesoft BASIC Programmer's Reference Manual
- ProDOS 8 Technical Reference
- BASIC Programming with ProDOS
- Apple Numerics Manual
- ImageWriter II Technical Reference Manual



Apple® II



# Apple IIc Technical Reference Manual



**Addison-Wesley Publishing Company, Inc.**

Reading, Massachusetts   Menlo Park, California   Don Mills, Ontario  
Wokingham, England   Amsterdam   Bonn   Sydney   Singapore   Tokyo  
Madrid   Bogotá   Santiago   San Juan

🍏 APPLE COMPUTER, INC.

Copyright © 1984, 1986 by  
Apple Computer, Inc.

All rights reserved. No part of  
this publication may be  
reproduced, stored in a  
retrieval system, or  
transmitted, in any form or by  
any means, electronic,  
mechanical, photocopying,  
recording, or otherwise,  
without prior written permission  
of Apple Computer, Inc.  
Printed in the United States of  
America.

Apple, the Apple logo,  
ProDOS, and LaserWriter are  
registered trademarks of Apple  
Computer, Inc.

Macintosh is a trademark of  
Apple Computer, Inc.

Microsoft is a registered trade-  
mark of Microsoft Corporation.

POSTSCRIPT is a trademark of  
Adobe Systems Incorporated.

ITC Garamond, ITC Avant  
Garde Gothic, and ITC Zapf  
Dingbats are registered  
trademarks of International  
Typeface Corporation.

Simultaneously published in the  
United States and Canada.

ISBN 0-201-17752-8  
ABCDEFGHIJ-DO-89876  
First printing, March 1987

## WARRANTY INFORMATION

**ALL IMPLIED WARRANTIES ON  
THIS MANUAL, INCLUDING  
IMPLIED WARRANTIES OF  
MERCHANTABILITY AND  
FITNESS FOR A PARTICULAR  
PURPOSE, ARE LIMITED IN  
DURATION TO NINETY (90)  
DAYS FROM THE DATE OF THE  
ORIGINAL RETAIL PURCHASE  
OF THIS PRODUCT.**

Even though Apple has reviewed  
this manual, **APPLE MAKES NO  
WARRANTY OR REPRESENTA-  
TION, EITHER EXPRESS OR  
IMPLIED, WITH RESPECT TO  
THIS MANUAL, ITS QUALITY,  
ACCURACY, MERCHANTABILITY,  
OR FITNESS FOR A PARTICULAR  
PURPOSE. AS A RESULT, THIS  
MANUAL IS SOLD "AS IS," AND  
YOU, THE PURCHASER, ARE  
ASSUMING THE ENTIRE RISK AS  
TO ITS QUALITY AND  
ACCURACY.**

**IN NO EVENT WILL APPLE BE  
LIABLE FOR DIRECT, INDIRECT,  
SPECIAL, INCIDENTAL, OR  
CONSEQUENTIAL DAMAGES  
RESULTING FROM ANY DEFECT  
OR INACCURACY IN THIS  
MANUAL, even if advised of the  
possibility of such damages.**

**THE WARRANTY AND REMEDIES  
SET FORTH ABOVE ARE EXCLU-  
SIVE AND IN LIEU OF ALL  
OTHERS, ORAL OR WRITTEN,  
EXPRESS OR IMPLIED.** No Apple  
dealer, agent, or employee is  
authorized to make any modifica-  
tion, extension, or addition to this  
warranty.

Some states do not allow the exclu-  
sion or limitation of implied warran-  
ties or liability for incidental or  
consequential damages, so the  
above limitation or exclusion may  
not apply to you. This warranty  
gives you specific legal rights, and  
you may also have other rights  
which vary from state to state.



## **Figures and tables xiv**

### **Preface About This Manual xxi**

Contents of this manual	xxi
The Apple IIc family	xxiii
Identifying your Apple IIc	xxiii
The original Apple IIc	xxiv
The UniDisk 3.5 Apple IIc	xxiv
The memory expansion Apple IIc	xxiv
Conventions used in this manual	xxv

### **Chapter 1 Introduction 1**

The outside of the machine	2
The keyboard	3
Features	3
Special function keys	4
Cursor movement keys	4
Modifier keys	5
The 80/40 switch	5
The keyboard switch	6
Disk-use and power lights	7
The speaker	8
The built-in disk drive	8
The back panel	9
The inside of the machine	11
The internal voltage converter	11
The main logic board	12
The other circuit boards	15

## **Chapter 2**   **Memory Organization and Control**   17

The 65C02 microprocessor	18
Overview of the address space	20
Memory map and memory switching	20
Main RAM addresses (\$0000–\$BFFF and \$D000–\$FFFF)	22
Auxiliary RAM addresses (\$0000–BFFF and \$D000–\$FFFF)	22
ROM addresses (\$C100–\$FFFF)	22
Hardware addresses (\$C000–\$C0FF)	23
Bank-switched memory	24
Page allocations	26
Page \$00 (one-byte addresses)	26
Page \$01 (the 65C02 stack)	26
Pages \$D0–\$FF (ROM and RAM)	26
Using bank selector switches	27
48K memory	36
Page allocations	36
Page \$02 (the input buffer)	36
Page \$03 (global storage and vectors)	36
Pages \$04–\$07 (text and low-resolution Page 1)	36
Pages \$08–\$0B (text and low-resolution Page 2)	38
Pages \$08 (communication port buffers)	38
Pages \$20–\$3F (high-resolution Page 1)	38
Pages \$40–\$5F (high-resolution Page 2)	39
Using 48K memory switches	39
Transfers between main and auxiliary memory	42
Transferring data	42
Transferring control	43
Using display memory switches	44
The reset routine	49
The cold-start procedure (power on)	51
The warm-start procedure (Control-Reset)	51
Forced cold start (Open Apple-Control-Reset)	52
The reset vector	52

## **Chapter 3**   **Introduction to Apple IIc I/O**   55

The standard I/O links	56
Standard input features	58
RdKey subroutine	58
KeyIn subroutine	58
GetLn subroutine	59
Escape codes with GetLn	60
Editing with GetLn	63
Cancel line	63
Backspace	63
Retype	63

Standard output features	64
COut subroutine	64
Control characters with COut1	65
Control characters with C3COut1	65
The stop-list feature	67
The text window	68
Normal, inverse, and flashing text	69
Primary character set display	70
Alternate character set display	70
Port I/O	71
Standard link entry points	71
Firmware protocol	72
Port I/O space	73
Port ROM space	73
Expansion ROM space	74
Port screen hole RAM space	74
Interrupts	75

## **Chapter 4 Keyboard and Speaker 77**

Keyboard input	78
Reading the keyboard	78
Monitor firmware support for keyboard input	82
Speaker output	82
Using the speaker	83
Monitor firmware support for speaker output	84

## **Chapter 5 Video Display Output 85**

Video display specifications	87
Text modes	88
Text character sets	88
MouseText	90
40-column versus 80-column text	91
Graphics modes	94
Low-resolution graphics	94
High-resolution graphics	95
Double high-resolution graphics	97
Mixed-mode displays	98
Display pages	99
Display mode switching	101
Display page maps	105
Monitor support for video display output	112
I/O firmware support for video display output	116

## **Chapter 6**   **Block Device I/O**   119

Disk drive I/O	120
Startup	121
Cold start	121
Warm start	123
Memory expansion card I/O	123
The Smartport I/O interface	123
Locating the Smartport	124
Issuing a call to the Smartport	125
Cautions	126
Descriptions of the Smartport calls	126
STATUS	128
Parameter descriptions	128
Possible errors	132
READ BLOCK	132
Parameter descriptions	133
Possible errors	133
WRITE BLOCK	134
Parameter descriptions	134
Possible errors	135
FORMAT	135
Parameter descriptions	135
Possible errors	136
CONTROL	136
Parameter descriptions	136
Possible errors	139
INIT	139
Parameter descriptions	140
Possible errors	140
OPEN	140
Parameter descriptions	140
Possible errors	141
CLOSE	141
Parameter descriptions	141
Possible errors	142
READ	142
Parameter descriptions	142
Possible errors	143
WRITE	143
Parameter descriptions	144
Possible errors	144
An example: issuing a Smartport call	145
Summary of commands and parameters	149
Summary of error codes	150

## **Chapter 7 Serial I/O Port 1 153**

- Using serial port 1 155
- Characteristics of port 1 at startup 159
- Hardware page locations for port 1 159
- I/O firmware support for port 1 160
- Screen hole locations for port 1 160
- Changing port 1 characteristics 161
  - Data format and baud rate 163
  - Carriage return and line feed 164
  - Sending special characters 165
  - Displaying output on the screen 165

## **Chapter 8 Serial I/O Port 2 167**

- Using serial port 2 169
- Characteristics of port 2 at startup 173
- Hardware page locations for port 2 173
- I/O firmware support for port 2 174
- Screen hole locations for port 2 174
- Changing port 2 characteristics 176
  - Data format and baud rate 177
  - Carriage return and line feed 179
  - Routing input and output 179
    - Half-duplex operation 180
    - Full-duplex operation 182
  - Terminal mode 184

## **Chapter 9 Mouse and Game Input 185**

- Mouse input 186
  - Mouse connector signals 187
  - Mouse operating modes 187
    - Transparent mode 187
    - Movement interrupt mode 187
    - Button interrupt mode 188
    - Movement/button interrupt mode 188
    - Vertical blanking active modes 188
  - Mouse soft switches 189
  - I/O firmware support for mouse input 191
    - Pascal support 195
    - BASIC and assembly-language support 195
  - Screen holes 196
  - Using the mouse as a hand controller 198

- Game input 198
  - The hand controller connector signals 199
  - Switch inputs (Sw0 and Sw1) 200
  - Analog inputs (Pdl0 and Pdl1) 200
  - Monitor support for game input 201

## **Chapter 10 Using the Monitor 203**

- Invoking the Monitor 204
- Syntax of Monitor commands 205
- Monitor memory commands 205
  - Examining memory contents 206
  - Memory dump 206
  - Changing memory contents 208
    - Changing one byte 208
    - Changing consecutive locations 209
  - Moving data in memory 210
  - Comparing data in memory 211
- Monitor register commands 212
  - Changing registers 213
  - Examining registers 213
- Miscellaneous Monitor commands 213
  - Display inverse and normal 214
  - Back to BASIC 214
  - Redirecting input and output 215
  - Hexadecimal arithmetic 215
- Advanced operations 216
  - Multiple-command lines 216
  - Filling memory 216
  - Repeating commands 217
  - Creating your own commands 218
- Machine-language programs 219
  - Running a program 219
  - Disassembled programs 220
- The STEP and TRACE commands 221
- The Mini-Assembler 223
  - Starting the Mini-Assembler 223
  - Using the Mini-Assembler 224
  - Mini-Assembler instruction formats 226
- Summary of Monitor commands 227
  - Examining memory 227
  - Changing the contents of memory 227
  - Moving and comparing 227
  - The Register command 228
  - Miscellaneous Monitor commands 228
  - Running and listing programs 229

<b>Chapter 11</b>	<b>Hardware Implementation</b>	<b>231</b>
	Environmental specifications	232
	Power requirements	233
	The external power supply	233
	The external power connector	234
	The internal converter	234
	Apple IIc overall block diagram	235
	The 65C02 microprocessor	237
	65C02 block diagram	237
	65C02 timing	239
	The custom integrated circuits	241
	The memory management unit (MMU)	241
	The input/output unit (IOU)	243
	The timing generator (TMG)	245
	The general logic unit (GLU)	245
	The disk controller unit (IWM)	247
	Memory addressing	248
	ROM addressing	249
	RAM addressing	251
	Dynamic RAM refreshment	251
	Dynamic RAM timing	252
	The keyboard	254
	The speaker	256
	Volume control	256
	Output jack	256
	The video display	257
	The video counters	257
	Display memory addressing	258
	Display address mapping	258
	Video display modes	261
	Text displays	263
	Low-resolution display	266
	High-resolution display	267
	Double high-resolution display	269
	Video output signals	270
	Monitor output	270
	Video expansion output	271
	Disk I/O	273
	Serial I/O	274
	ACIA control register	278
	ACIA command register	280
	ACIA status register	281
	ACIA transmit/receive register	282

- Mouse input 282
- Hand controller input 287
- Memory expansion card 291
- Schematic diagrams 291

## **Appendix A The 65C02 Microprocessor 297**

- Differences between 6502 and 65C02 297
  - Differing cycle times 297
  - Differing instruction results 298
- Data sheet 298

## **Appendix B Memory Map 308**

- Page \$00 308
- Page \$03 312
- Screen holes 312
- The hardware page 316

## **Appendix C Important Firmware Locations 322**

- The tables 322
- Port addresses 323
- Other video and I/O firmware addresses 326
- Applesoft BASIC interpreter addresses 326
- Monitor addresses 326

## **Appendix D Operating Systems and Languages 328**

- Operating systems 328
  - ProDOS 328
  - DOS 328
  - Pascal Operating System 329
- Languages 329
  - Applesoft BASIC 329
  - Integer BASIC 330
  - Pascal 330
  - Fortran 330
  - Logo II 330

## **Appendix E Interrupts 331**

- Introduction 331
  - What is an interrupt? 331
  - Interrupts on Apple II computers 332
  - Interrupt handling on the 65C02 333
  - The interrupt vector at \$FFFE 333

The built-in interrupt handler	334
Saving the memory configuration	335
Managing main and auxiliary stacks	336
User's interrupt handler at \$03FE	336
Handling break instructions	337
Sources of interrupts	338
Firmware handling of interrupts	339
Firmware for mouse and VBL	339
Firmware for keyboard interrupts	340
Using keyboard buffering firmware	341
Using keyboard interrupts through firmware	342
Using external interrupts through firmware	342
Firmware for serial interrupts	343
Using serial buffering transparently	343
Using serial interrupts through firmware	344
Transmitting serial data	344
A loophole in the firmware	345
Bypassing the interrupt firmware	345
Using mouse interrupts without the firmware	345
Using ACIA interrupts without the firmware	347

## **Appendix F Apple II Series Differences 348**

Overview	348
Type of processor	350
Machine identification	350
Memory structure	351
Amount and address ranges of RAM	351
Amount and address ranges of ROM	351
Peripheral-card memory spaces	352
Hardware addresses	353
\$C000-\$C00F	353
\$C010-\$C01F	353
\$C020-\$C02F	354
\$C030-\$C03F	354
\$C040-\$C04F	354
\$C050-\$C05F	354
\$C060-\$C06F	355
\$C070-\$C07F	355
\$C080-\$C08F	356
\$C090-\$C0FF	356
Monitors	356

I/O in general	357
DMA transfers	357
Slots versus ports	357
Interrupts	357
The keyboard	357
Keys, switches, and lights	358
Character sets	358
The speaker	359
The video display	359
Character sets	359
MouseText	360
Vertical blanking	360
Display modes	360
Disk I/O	361
Serial I/O	361
Serial ports versus serial cards	361
Serial I/O buffers	362
Mouse and hand controllers	363
Mouse input	363
Hand controller input and output	363
Cassette I/O	364
Hardware	365
Power	365
Custom chips	365

## **Appendix G    USA and International Models 366**

Keyboard layouts and codes	366
USA standard (Sholes) keyboard	367
USA simplified (Dvorak) keyboard	370
ISO layout of USA keyboard	371
English keyboard	372
French keyboard	373
Canadian keyboard	375
German keyboard	376
Italian keyboard	378
Western Spanish keyboard	380
ASCII character sets	381
Certification	383
Product safety	383
Important safety instructions	383
Power supply specifications	383

**Appendix H   Conversion Tables   384**

Bits and bytes   384  
Hexadecimal and decimal   387  
Hexadecimal and negative decimal   388  
Peripheral identification numbers   389  
Eight-bit code conversions   391

**Appendix I   Firmware Listings   396**

**Glossary   509**  
**Bibliography   533**  
**Index   535**  
**Tell Apple Card**

---

---

## Figures and tables

### Chapter 1 Introduction 1

Figure 1-1	Apple IIc external features, front	2
Figure 1-2	Apple IIc external features, back	2
Figure 1-3	Front of Apple IIc with standard USA keyboard	3
Figure 1-4	USA standard (or Sholes) keyboard, keyboard switch up	6
Figure 1-5	USA simplified (or Dvorak) keyboard, keyboard switch down	7
Figure 1-6	Speaker, volume control, and audio output jack	8
Figure 1-7	Built-in disk drive	9
Figure 1-8	Back panel connectors	10
Figure 1-9	Inside the machine	11
Figure 1-10	Power supply and voltage converter	12
Figure 1-11	Original and UniDisk 3.5 IIc main logic board	13
Figure 1-12	Memory expansion IIc main logic board	14
Table 1-1	Keyboard specifications	

### Chapter 2 Memory Organization and Control 17

Figure 2-1	Internal model of the 65C02 microprocessor	19
Figure 2-2	Apple IIc memory map	21
Figure 2-3	Bank-switched memory map	25
Figure 2-4	Read ROM	29
Figure 2-5	Read ROM, write RAM, and use first \$D0 bank	30
Figure 2-6	Read ROM, write RAM, and use second \$D0 bank	31
Figure 2-7	Read RAM and use first \$D0 bank	32
Figure 2-8	Read RAM and use second \$D0 bank	33
Figure 2-9	Read and write RAM and use first \$D0 bank	34
Figure 2-10	Read and write RAM and use second \$D0 bank	35
Figure 2-11	48K memory map	37
Figure 2-12	48K RAM selection, split pairs	40
Figure 2-13	48K RAM selection, one side only	41
Figure 2-14	Page2 selections, 80Store on and HiRes off	47
Figure 2-15	Page2 selections, 80Store on and HiRes on	48
Figure 2-16	Reset routine flowchart	49
Table 2-1	Bank selector switches	28

Table 2-2	48K memory switches	39
Table 2-3	48K RAM transfer routines	42
Table 2-4	Parameters for MoveAux routine	43
Table 2-5	Parameters for XFer routine	43
Table 2-6	Display memory switches	45
Table 2-7	Page \$03 vectors	50

### **Chapter 3 Introduction to Apple IIc I/O 55**

Table 3-1	Prompt characters	59
Table 3-2	Escape codes with GetLn	61
Table 3-3	Control characters with COut1	65
Table 3-4	Control characters with C3COut1	66
Table 3-5	Text window memory locations	69
Table 3-6	Port characteristics	71
Table 3-7	Firmware protocol locations	72
Table 3-8	Port I/O locations	73
Table 3-9	Port screen hole memory locations	74

### **Chapter 4 Keyboard and Speaker 77**

Table 4-1	Keyboard input characteristics	79
Table 4-2	Keys and ASCII codes	80
Table 4-3	Speaker output characteristics	83

### **Chapter 5 Video Display Output 85**

Figure 5-1	MouseText characters	91
Figure 5-2	40-column and 80-column text with alternate character set	92
Figure 5-3	Text mode characteristics and switching	93
Figure 5-4	High-resolution display bits	96
Figure 5-5	Map of 40-column text display	107
Figure 5-6	Map of 80-column text display	108
Figure 5-7	Map of low-resolution graphics display	109
Figure 5-8	Map of high-resolution graphics display	110
Figure 5-9	Map of double high-resolution graphics display	111
Table 5-1	Video output port characteristics	86
Table 5-2	Video display specifications	87
Table 5-3	Display character sets	89
Table 5-4	Low-resolution graphics colors	94
Table 5-5	High-resolution graphics colors	97
Table 5-6	Double high-resolution graphics colors	99
Table 5-7	Video display page locations	101
Table 5-8	Display soft switches	102

Table 5-9	Display modes supported by firmware, including Applesoft 104
Table 5-10	Other display modes 104
Table 5-11	Monitor firmware routines 112
Table 5-12	Port 3 firmware protocol table 116
Table 5-13	Pascal video control functions 117

## **Chapter 6 Block Device I/O 119**

Figure 6-1	Summary of Smartport calls 149
Table 6-1	Disk I/O port characteristics 120

## **Chapter 7 Serial I/O Port 1 153**

Figure 7-1	Diagram of port 1 characteristics storage 162
Figure 7-2	Data format 163
Table 7-1	Serial port 1 characteristics 154
Table 7-2	Printer port commands 155
Table 7-3	Port 1 hardware page locations 159
Table 7-4	Port 1 I/O firmware protocol 160
Table 7-5	Port 1 screen hole locations 160

## **Chapter 8 Serial I/O Port 2 167**

Figure 8-1	Diagram of port 2 characteristics storage 177
Figure 8-2	Devices in a typical communication setup 178
Figure 8-3	Effect of IN#2 180
Figure 8-4	Effect of IN#2 and T command, half duplex 181
Figure 8-5	Effect of IN#2 and T command, full-duplex terminal 182
Figure 8-6	Effect of IN#2, PR#2, and T command, full-duplex host 183
Table 8-1	Serial port 2 characteristics 168
Table 8-2	Modem port commands 170
Table 8-3	Port 2 hardware page locations 174
Table 8-4	Port 2 I/O firmware protocol 174
Table 8-5	Port 2 screen hole locations 175

## **Chapter 9 Mouse and Game Input 185**

Table 9-1	Mouse input port characteristics 186
Table 9-2	Mouse soft switches 189
Table 9-3	Mouse firmware routines 193
Table 9-4	Mouse port I/O firmware protocol 195
Table 9-5	Mouse port screen hole locations 197
Table 9-6	Game input characteristics 199

## **Chapter 10 Using the Monitor 203**

Table 10-1	Mini-Assembler address formats	226
------------	--------------------------------	-----

## **Chapter 11 Hardware Implementation 231**

Figure 11-1	External power connector	234
Figure 11-2	Apple IIc block diagram	236
Figure 11-3	65C02 block diagram	238
Figure 11-4	65C02 timing signals	240
Figure 11-5	MMU pinouts	242
Figure 11-6	IOU pinouts	243
Figure 11-7	TMG pinouts	245
Figure 11-8	GLU pinouts	246
Figure 11-9	IWM pinouts	247
Figure 11-10	Memory bus organization	249
Figure 11-11	23128 ROM pinouts	249
Figure 11-12	2316 ROM pinouts	250
Figure 11-13	2364 pinouts	250
Figure 11-14	64K RAM pinouts	251
Figure 11-15	RAM timing signals	253
Figure 11-16	Keyboard circuit diagram	254
Figure 11-17	Keyboard signals	255
Figure 11-18	Speaker circuit diagram	256
Figure 11-19	Display address transformation	260
Figure 11-20	40-column text display memory	261
Figure 11-21	Video display circuits	262
Figure 11-22	7-MHz video timing signals: 40-column, low-resolution, and high-resolution display	264
Figure 11-23	14-MHz video timing signals: 80-column and double high-resolution display	265
Figure 11-24	Video output back panel connectors	270
Figure 11-25	Video expansion connector pinouts	272
Figure 11-26	Disk drive connector	274
Figure 11-27	Serial port circuits	275
Figure 11-28	6551 ACIA block diagram	276
Figure 11-29	6551 pinouts	277
Figure 11-30	Serial port connectors	278
Figure 11-31	ACIA control register	279
Figure 11-32	ACIA command register	280
Figure 11-33	ACIA status register	281
Figure 11-34	Sample mouse waveform	283
Figure 11-35	Mouse movement and direction waveforms	283
Figure 11-36	Mouse connector	284
Figure 11-37	Mouse circuits	285
Figure 11-38	Mouse button signals	286

Figure 11-39	Hand controller connector	287
Figure 11-40	How to connect switch inputs	288
Figure 11-41	Hand controller circuits	288
Figure 11-42	Hand controller signals	289
Figure 11-43	Memory expansion card connector pinout diagram	291
Figure 11-44	Apple IIc schematic diagram	292
Table 11-1	Environmental specifications	232
Table 11-2	Power supply specifications	233
Table 11-3	External power connector signals	234
Table 11-4	Internal converter specifications	234
Table 11-5	65C02 microprocessor specifications	239
Table 11-6	65C02 timing signal descriptions	240
Table 11-7	MMU signal descriptions	242
Table 11-8	IOU signal descriptions	243
Table 11-9	TMG signal descriptions	245
Table 11-10	GLU signal descriptions	246
Table 11-11	IWM signal descriptions	247
Table 11-12	RAM address multiplexing	252
Table 11-13	RAM timing signals	253
Table 11-14	Display memory addressing	260
Table 11-15	Memory address bits for display modes	260
Table 11-16	Character-generator control signals	266
Table 11-17	Video expansion connector signals	272
Table 11-18	Disk drive connector signals	274
Table 11-19	6551 signal descriptions	277
Table 11-20	Serial port connector signals	278
Table 11-21	Mouse connector signals	284
Table 11-22	Hand controller connector signals	287

## **Appendix A The 65C02 Microprocessor 297**

Table A-1	Cycle time differences	298
-----------	------------------------	-----

## **Appendix B Memory Map 308**

Table B-1	Page \$00 use	309
Table B-2	Page \$03 use	312
Table B-3	Main memory screen hole allocations	313
Table B-4	Auxiliary memory screen hole allocations	315
Table B-5	Addresses \$C000–\$C03F	316
Table B-6	Addresses \$C040–\$C05F	318
Table B-7	Addresses \$C060–\$C07F	319
Table B-8	Addresses \$C080–\$C0AF	320
Table B-9	Addresses \$C0B0–\$C0FF	321

## **Appendix C Important Firmware Locations 322**

Table C-1	Serial port 1 addresses	323
Table C-2	Serial port 2 addresses	324
Table C-3	Video firmware addresses	324
Table C-4	Mouse port addresses	325
Table C-5	Apple IIc enhanced video and miscellaneous firmware	326
Table C-6	Apple IIc monitor entry points and vectors	326

## **Appendix E Interrupts 331**

Table E-1	Interrupt-handling sequence	335
Table E-2	Activating mouse interrupts	346
Table E-3	Reading mouse interrupts	346

## **Appendix F Apple II Series Differences 348**

Figure F-1	Apple II, II Plus, and IIe hand controller signals	364
Table F-1	Apple II series identification bytes	350

## **Appendix G USA and International Models 366**

Figure G-1	USA standard (or Sholes) keyboard, keyboard switch up	368
Figure G-2	USA simplified (or Dvorak) keyboard, keyboard switch down	370
Figure G-3	ISO version of USA standard keyboard, keyboard switch up	371
Figure G-4	English keyboard, keyboard switch up	372
Figure G-5	French keyboard, keyboard switch down	373
Figure G-6	Canadian keyboard, keyboard switch down	375
Figure G-7	German keyboard, keyboard switch down	376
Figure G-8	Italian keyboard, keyboard switch down	378
Figure G-9	Western Spanish keyboard, keyboard switch down	380
Table G-1	Keys and ASCII codes	368
Table G-2	English keyboard code differences from Table G-1	372
Table G-3	French keyboard code differences from Table G-1	374
Table G-4	Canadian keyboard code differences from Table G-1	375
Table G-5	German keyboard code differences from Table G-1	377

Table G-6	Italian keyboard code differences from Table G-1	379
Table G-7	Western Spanish keyboard code differences from Table G-1	381
Table G-8	ASCII code equivalents	381
Table G-9	50-Hz power supply specifications	383

## **Appendix H    Conversion Tables    384**

Figure H-1	Bits, nibbles, and bytes	386
Table H-1	What a bit can represent	385
Table H-2	Values represented by a nibble	386
Table H-3	Hexadecimal/decimal conversion	387
Table H-4	Hexadecimal to negative decimal conversion	388
Table H-5	PIN numbers	390
Table H-6	Control characters, high bit off	392
Table H-7	Special characters, high bit off	393
Table H-8	Uppercase characters, high bit off	394
Table H-9	Lowercase characters, high bit off	395

## **Appendix I    Firmware Listings    396**

Table I-1	Main side ROM map	397
Table I-2	Auxiliary side ROM map	398



# Preface



## About This Manual

This is the reference manual for the Apple® IIc personal computer. It contains detailed descriptions of all the hardware and firmware that make up the Apple IIc and provides the technical information that peripheral-card designers and programmers need.

The information in this manual is aimed at assembly-language programmers and hardware designers, but others interested in the internal operation of the Apple IIc can also benefit from reading it.

This manual tells you how the Apple IIc works, but not how to use it. If you need to know how to set up and use your Apple IIc, read the *Apple IIc Owner's Manual*.

This manual describes three versions of the Apple IIc:

- the original Apple IIc
- the Apple IIc that supports the UniDisk™ 3.5 drive
- the Apple IIc that supports the Memory Expansion Card

More information on the various versions of the Apple IIc is provided under "The Apple IIc Family," later in this Preface.

---

---

## Contents of this manual

The Apple IIc is presented in this manual from the outside in.

Chapter 1 introduces the Apple IIc, including external controls, connectors, and the main internal components.

Chapter 2 introduces the 65C02 microprocessor and its directly addressable memory space.

Chapter 3 introduces the I/O characteristics of the Apple IIc. Chapters 4 and 9 cover specific areas of the I/O interface.

Chapter 4 describes the keyboard and speaker.

Chapter 5 describes the video display.

Chapter 6 describes block device I/O, including the Smartport firmware interface.

Chapter 7 describes serial port 1.

Chapter 8 describes serial port 2.

Chapter 9 describes the mouse/game paddle port.

Chapter 10 describes the Apple IIc's built-in Monitor firmware. The Monitor helps you write, disassemble, and debug machine-language programs, as well as providing you with a means to look at and manipulate the contents of main memory.

Chapter 11 describes the Apple IIc hardware in detail.

Appendix A describes the 65C02 microprocessor in detail, including the differences between it and the 6502 microprocessor used on early-model Apple II's. Most of this appendix is a reprint of the manufacturer's data sheet for the 65C02.

Appendix B contains a memory map of the Apple IIc main memory. Detailed maps are provided for memory pages \$00 and \$03, the screen holes, and the hardware page.

Appendix C lists the Apple IIc firmware entry points, including those for the I/O firmware and the Monitor firmware.

Appendix D describes some of the operating systems and languages supported by Apple Computer for the Apple IIc.

Appendix E describes the operation of the Apple IIc interrupt handler firmware and how to use it in your programs.

Appendix F outlines the differences and similarities between the diverse members of the Apple II family of computers.

Appendix G describes the various international versions of the Apple IIc keyboard and character set. Power and safety information for international versions of the Apple IIc is also included in this appendix.

Appendix H contains tables to aid you in code and number base conversions.

Appendix I contains the firmware listing for the new version of the Apple IIc and information on obtaining listings for the original and UniDisk 3.5 ROMs.

The Glossary defines many of the technical terms used in this manual.

The Bibliography lists articles and books with additional information about the Apple IIc.

Finally, after the index at the back of this manual, you'll find the Tell Apple Card; please take a minute to fill this card out and mail it back to us. Your experience with this and other Apple manuals can help us plan new reference materials.

---

---

## The Apple IIc family

Changes have been made to the Apple IIc since the original version was introduced. The first change was made in order to support the UniDisk 3.5 external drive, and included a set of ROM-based machine-language routines called the **Protocol Converter**. The latest version incorporates all the UniDisk 3.5 upgrade features, a new version of the Protocol Converter called the **Smartport**, and support for an optional memory expansion card. All of these versions are described in this manual. Where there are differences between the various versions of the Apple IIc, they will be called out in the manual. For the sake of convenience, the various versions of the Apple IIc are identified by the features they support, such as *memory expansion* for the newest IIc and *UniDisk 3.5* for the version that introduced the UniDisk 3.5 drive support. Unless specified, all versions of the Apple IIc operate identically.

### Important

Smartport is merely a new name for the Protocol Converter; all the specifications for the Smartport apply to the Protocol Converter, and vice versa.

---

---

## Identifying your Apple IIc

There are basically three versions of the Apple IIc:

- ☐ the *original* Apple IIc
- ☐ the *UniDisk 3.5* Apple IIc
- ☐ the *memory expansion* Apple IIc

You can tell which Apple IIc you have by checking the value of the ID byte at ROM location 64447 (\$FBBF in hexadecimal). The value of this byte is 255 (\$FF) in the original Apple IIc, 0 (\$00) in the UniDisk 3.5 version, and 3 (\$03) in the memory expansion version.

❖ *Checking the ID byte:* You can check the value of the ID byte from Applesoft by typing `PRINT PEEK (64447)`.

---

## The original Apple IIc

The original Apple IIc is the oldest member of the IIc family. It has the following features:

- the 65C02 microprocessor
- 128K of RAM

---

## The UniDisk 3.5 Apple IIc

The Apple IIc that introduced support for the UniDisk 3.5 drive is identified in this manual as the UniDisk 3.5 version. It includes the following changes from the original Apple IIc:

- the Protocol Converter, to support the UniDisk 3.5 external disk drive
- a 256K ROM IC to replace the 128K ROM
- some new serial port commands
- the Mini-Assembler
- two new Monitor commands (STEP and TRACE)
- built-in diagnostics

The UniDisk 3.5 Apple IIc also includes improved interrupt handler features and new external drive startup procedures.

---

## The memory expansion Apple IIc

The Apple IIc that supports an optional memory expansion card supports all the features of the UniDisk 3.5 version. It includes the following changes from the UniDisk 3.5 IIc:

- an internal connector to support an optional memory expansion card
- 4 64Kx4 RAM ICs to replace the 16 64Kx1 ICs

The Apple IIc that supports the memory expansion option also reorganizes the I/O port ("slot") entry points in the firmware. The mouse, located at port 4 in the original and UniDisk 3.5 versions, is now at port 7. The memory expansion card uses port 4 in the new Apple IIc. What this means is that *all the mouse I/O entry point addresses have been changed from \$C4XX to \$C7XX.*

To avoid confusion and maintain compatibility with previous versions, the text and tables in this book still show the values used for the original and UniDisk 3.5 versions of the Apple IIc. However, a statement reminding you of the change appears near affected tables.

---

Remember that the Smartport and the Protocol Converter are the same thing.

---

---

---

## Conventions used in this manual

Special text in this manual is set off in several different ways, as shown in these examples.

---

Warning	Important warnings look like this. These flag potential danger to the Apple IIc, its software, or you.
---------	--

---

---

Important	Text set off like this is less urgent or threatening than text in a Warning box, but still of a critical nature.
-----------	--

---

---

Original IIc	Text set off like this applies only to the original version of the Apple IIc.
--------------	---

---

---

UniDisk 3.5	Text set off like this applies only to the UniDisk 3.5 version of the Apple IIc.
-------------	--

---

---

Memory expansion	Text set off like this applies only to the memory expansion version of the Apple IIc.
------------------	---

---

❖ *By the way:* Information that is useful but incidental to the text is set off like this. You may want to skip over such information and return to it later.

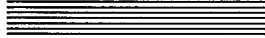
Terms that appear in **boldface** in the text are defined in the Glossary or a marginal gloss.

Computer voice is used to indicate text that should be identical to your screen display or printout.





# **Chapter 1**



## **Introduction**

This chapter introduces you to the working parts of the Apple IIc by briefly describing the major components of the computer—both internal and external hardware and firmware—and telling you where in the manual to find out more about them.

---

---

## The outside of the machine

This section briefly describes the Apple IIc's keyboard, controls, indicators, and expansion connectors.

The Apple IIc comes equipped with a keyboard, speaker (with audio output jack and volume control), built-in disk drive, external power supply, and internal voltage converter. It also has built-in interfaces with external connectors for a serial printer, video monitor, special video display adapters, modem, mouse, and game controllers. These external connectors allow you to plug in accessory equipment without having to go inside the machine to use expansion slots like those in the Apple IIe.

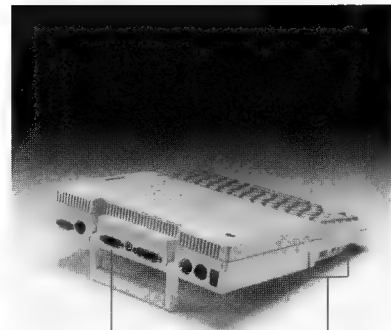
Figure 1-1 shows the front and right side of an Apple IIc, and Figure 1-2 shows the back and left side.



Keyboard  
(See Figs. 1-4 and 1-5)

Disk Drive  
(See Fig. 1-7)

**Figure 1-1**  
Apple IIc external features,  
front



Back Panel  
(See Fig. 1-8)

Speaker  
Volume Control  
(See Fig. 1-6)

**Figure 1-2**  
Apple IIc external features,  
back

---

## The keyboard

**ASCII** stands for *American Standard Code for Information Interchange*. Table 4-2 lists the ASCII character encoding for the standard and simplified USA keyboards. Appendix G lists the encoding for international keyboards.

The Apple IIc's primary input device is the keyboard, shown in Figure 1-3. The keyboard has a 63-key typewriter layout with both uppercase and lowercase characters and can generate all 128 standard **ASCII** characters. A reset key, 80/40-column display selector switch, keyboard layout selector switch, disk-use light, and power light are also located on the front of the computer.



**Figure 1-3**  
Front of Apple IIc with standard USA keyboard

Table 1-1 lists the characteristics of all Apple IIc keyboards and front panels.

### Features

The Apple IIc keyboard has automatic repeat on all character keys. This means that if you hold the key down longer than about a second, the character it generates repeats until you let up the key. It also has two-key rollover, which means if you press a key before releasing the one you pressed before it, the second character enters the computer the same as though you had released the previous key first. (This is important for fast touch-typists.)

**Table 1-1**  
Keyboard specifications

<b>Number of keys</b>	63
<b>Character encoding</b>	ASCII
<b>Number of codes</b>	128
<b>Features</b>	Automatic repeat, two-key rollover
<b>Special function keys</b>	Reset, Open Apple, Solid Apple,
<b>Cursor movement keys</b>	Left Arrow, Right Arrow, Down Arrow, Up Arrow, Return, Delete, Tab
<b>Modifier keys</b>	Control, Shift, Caps Lock, Escape
<b>Front-panel switches</b>	80/40 switch, keyboard switch
<b>Front-panel lights</b>	Power light, disk-use light

### Special function keys

The Apple IIc keyboard has three special function keys: Reset, and two keys marked with apples—one outlined (Open Apple) and one filled in (Solid Apple).

Reset has a direct line to the 65C02 microprocessor's RESET signal line (see Chapter 11): holding down Control while pressing Reset causes the Apple IIc to restart processing with an internal firmware program that puts the machine in a known state (see Chapter 2).

You can restart the Apple IIc without turning the power off and back on again, by holding down both Control and Open Apple while pressing Reset. Restarting this way is less stressful to the Apple IIc's components than normal powerup.

### Cursor movement keys

The Apple IIc keyboard has four cursor movement keys with arrows marked on them: left, right, down, and up. Three other keys can also cause cursor movements: Return, Delete, and Tab. All seven of these keys generate ASCII control characters (see Table 4-2). It is up to the operating system or application program to interpret and act on the control codes that these keys generate.

The Open Apple and Solid Apple keys are connected to 1-bit addresses in memory, described in Chapter 9.

Chapter 2 describes the results of the various reset procedures.

The **Monitor** is a built-in program that performs some of the basic activities of the computer, such as retrieving and storing key codes as they come in, and clearing or updating the display screen.

## Modifier keys

Three special keys—Control, Shift, and Caps Lock—generate no codes when pressed by themselves, but change the codes generated by other keys they are pressed in combination with. A fourth key, **Escape**, generates a nonprinting control code that causes the **Monitor** to interpret certain subsequent keystrokes in a modified way.

- Control, when pressed in combination with letter keys or certain other keys, produces ASCII control characters. Most of the control characters are invisible most of the time.
- Shift works the same on the Apple IIc as on an ordinary typewriter: it selects uppercase letters and the upper characters on the keys.
- Caps Lock, in its down position, changes the letter keys to uppercase, but does not affect other keys.
- Escape is not a modifier key in the same sense as Control and Shift: you do not hold it down while pressing other keys. Rather, you press Escape and it generates the ASCII escape (ESC) control character (key code \$1B—see Table 4-2). When the Escape key is pressed, many programs—including the built-in Monitor program—then interpret other specific keys as designating an escape sequence.

## The 80/40 switch

The 80/40 switch lets you specify whether a program should display information in 40 or 80 columns per line. The switch indicates 40-column display when in its down position, and 80-column display when in its up position.

---

### Important

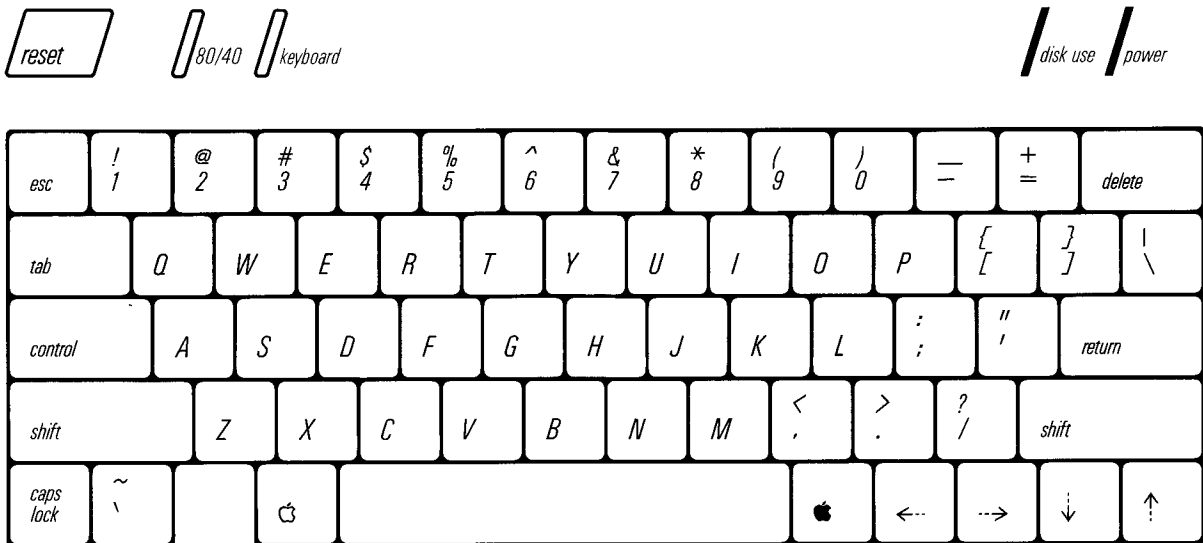
Not all programs check this switch. Even programs that do check the switch may do so only when the program first starts up. If that is the case, changing the switch position while the program is running will have no effect on the program's display. (See Table 4-1.)

---

## The keyboard switch

You use the keyboard switch to select for use one of the two keyboard layouts and screen character sets built into your Apple IIc. On USA versions of the Apple IIc, you select the standard Sholes keyboard layout (Figure 1-4) with the switch in the up position, and the Dvorak simplified layout (Figure 1-5) with the switch in the down position.

If you normally use the Dvorak keyboard layout, you can *gently* pry up the keys from the keyboard and rearrange and replace them in their Dvorak positions.



**Figure 1-4**  
USA standard (or Sholes) keyboard, keyboard switch up



**Figure 1-5**  
USA simplified (or Dvorak) keyboard, keyboard switch down (shaded characters may be in different positions on some models)

Appendix G illustrates the keyboard layouts for both keyboard switch positions on several international versions of the Apple IIc.

On international models, the keycaps indicate the character positions for the local keyboard layout, which is selected when the keyboard switch is down. When up, the keyboard switch selects the USA standard characters and key layout.

**Disk-use and power lights**

The red disk-use light glows whenever the built-in disk drive's motor is switched on.

The green power light glows when the Apple IIc is turned on and normal power is present at the Apple IIc's internal power supply.

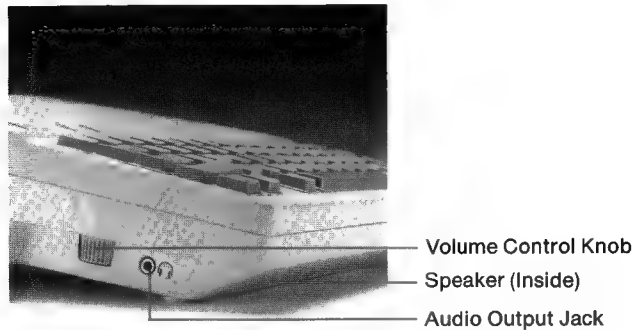
**Warning** If the power light flashes on and off, turn off the computer **immediately**. Find out what caused the condition (such as a brownout or short circuit) and fix the problem before turning the computer on again. Above all, do not use the disk drive when the power light is flashing; this may damage the computer.

The way programs control the speaker is described under "Speaker Output" in Chapter 4.

---

## The speaker

The Apple IIc has a speaker in the bottom of the case, as shown in Figure 1-6. The speaker lets Apple IIc programs produce a variety of sounds. There is also a volume control on the left side of the Apple IIc case, and a jack for connecting headphones or an external speaker. The jack accepts either one-channel (monaural) or two-channel (stereo) plugs, although speaker output is monaural only. Inserting a plug disconnects the built-in speaker

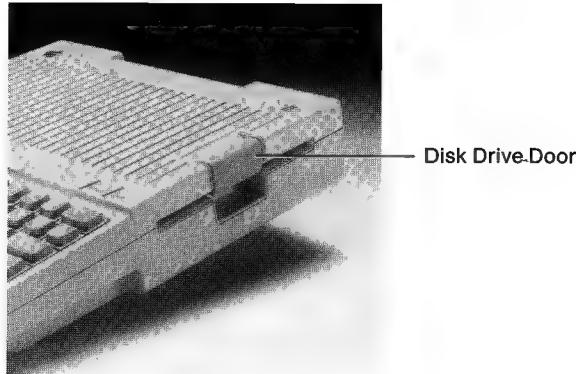


**Figure 1-6**  
Speaker, volume control, and audio output jack

---

## The built-in disk drive

The Apple IIc's built-in disk drive (Figure 1-7) is fully compatible with the Apple Disk IIc that reads and writes 5.25-inch single-sided 35-track disks. The drive door is on the right side of the Apple IIc case.



**Figure 1-7**  
Built-in disk drive

---

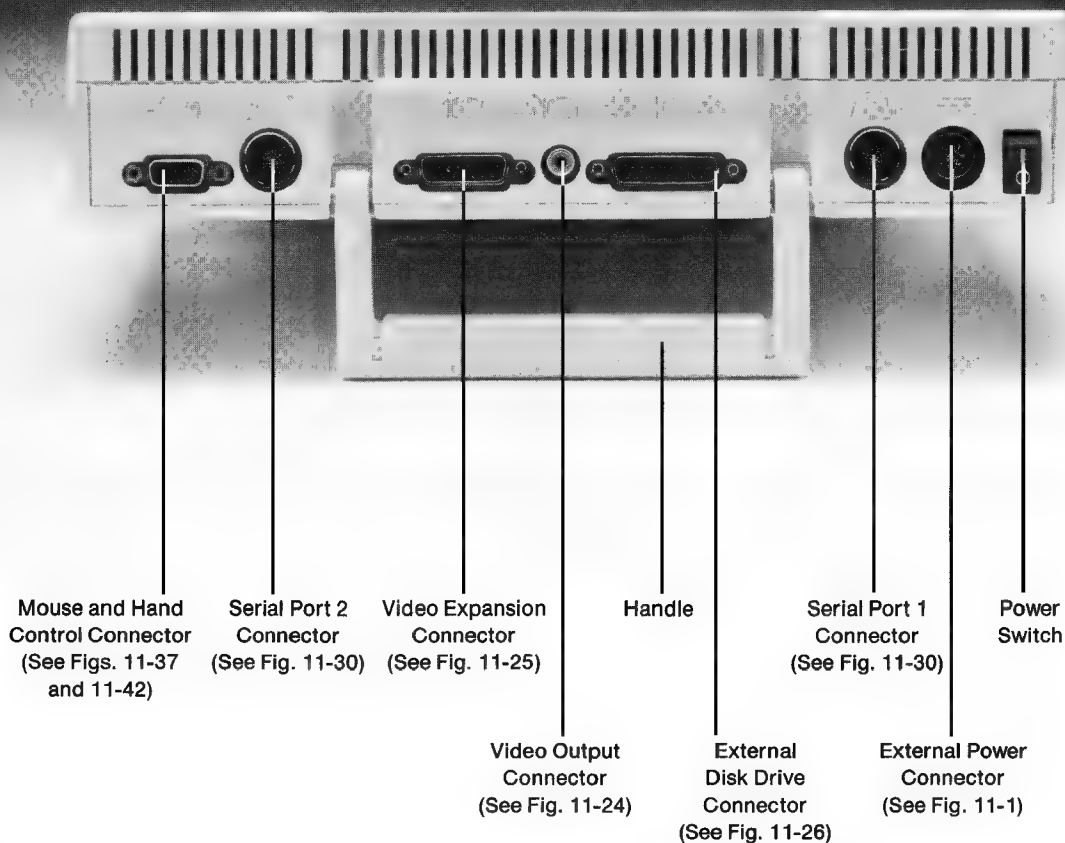
## The back panel

The back panel of the Apple IIc (Figure 1-8) has seven connectors and a main power switch. From left to right they are

- a 9-pin D-type miniature connector for connecting hand controllers, a mouse, a joystick, or some other device (see Chapters 9 and 11)
- a 5-pin DIN connector for serial input and output (port 2; normally for a modem) (see Chapters 7 and 11)
- a 15-pin D-type connector for video expansion (see Chapter 11)
- an RCA-type jack for a video monitor (see Chapter 11)
- a 19-pin D-type connector for connecting one or more external devices, such as intelligent disk drives (see Chapters 6 and 11)
- another 5-pin DIN connector for serial input and output (port 1; normally for a printer or plotter) (see Chapters 8 and 11)
- a special 7-pin DIN connector for power input (see Chapter 11)

Before attaching cables to the Apple IIc back panel connectors, be sure to move the handle until it clicks into position for propping up the computer. The handle should be down whenever the computer is running so that it can maintain proper cooling airflow.

The installation manuals for external devices contain instructions for connecting them to the Apple IIc.

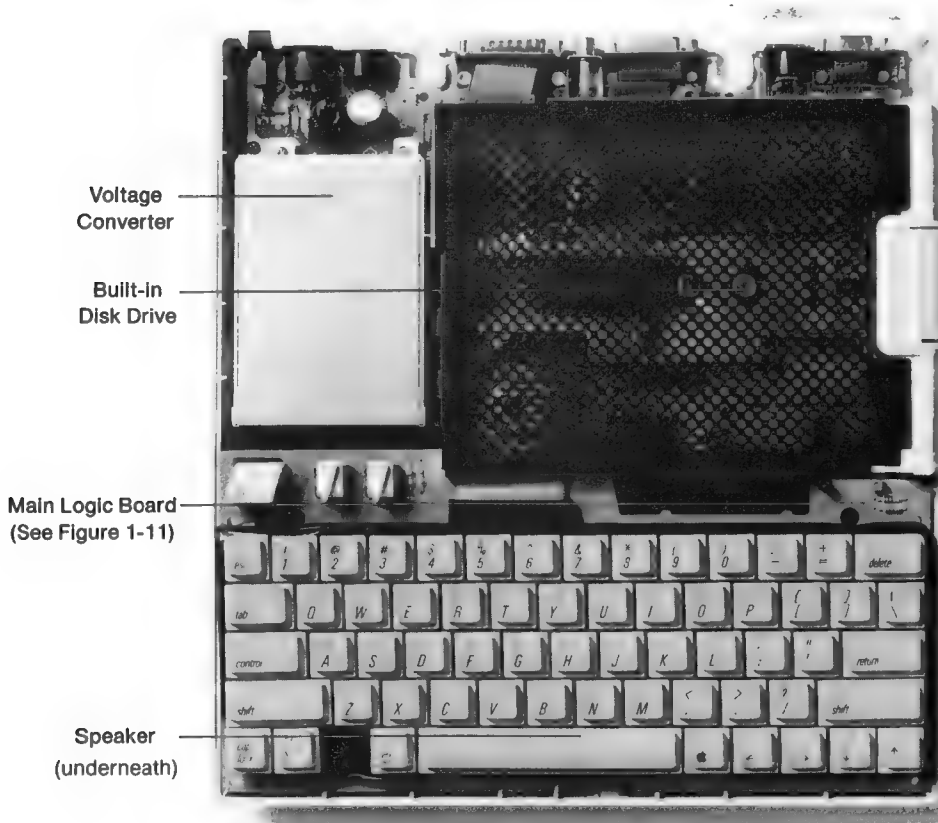


**Figure 1-8**  
Back panel connectors

---

## The inside of the machine

Figure 1-9 shows the main components inside the Apple IIc computer.



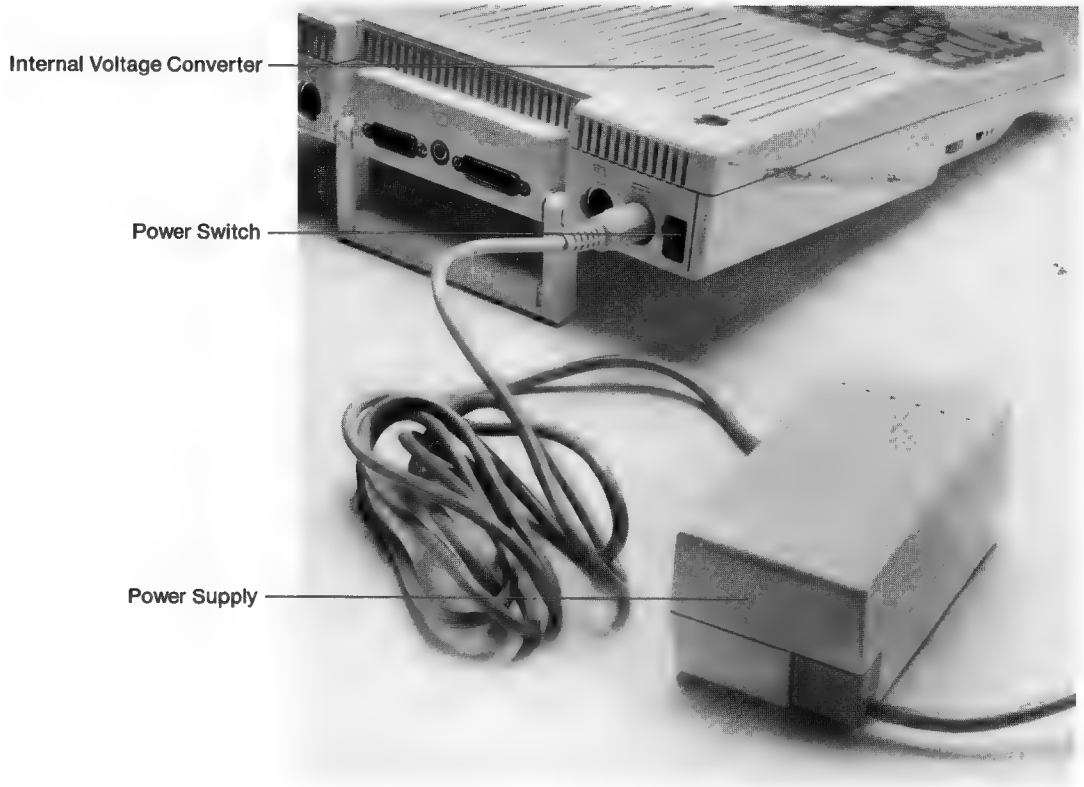
**Figure 1-9**  
Inside the machine

---

## The internal voltage converter

Complete specifications of the Apple IIc power supply and voltage converter appear in Chapter 11.

The built-in voltage converter operates from a 12 to 15 VDC input source, such as provided by the external power supply furnished with the Apple IIc (Figure 1-10). The voltage converter provides power for the logic board, built-in disk drive, one external disk drive, and the I/O signals available at the back panel.



**Figure 1-10**  
Power supply and voltage converter

The voltage converter produces three different voltages: +5V, +12V, and -12V. (Minus 5V, needed by some components in the Apple IIc, is derived from -12V on the main logic board.) It is a high-efficiency switching converter that protects itself and the rest of the Apple IIc against short circuits and other electrical mishaps.

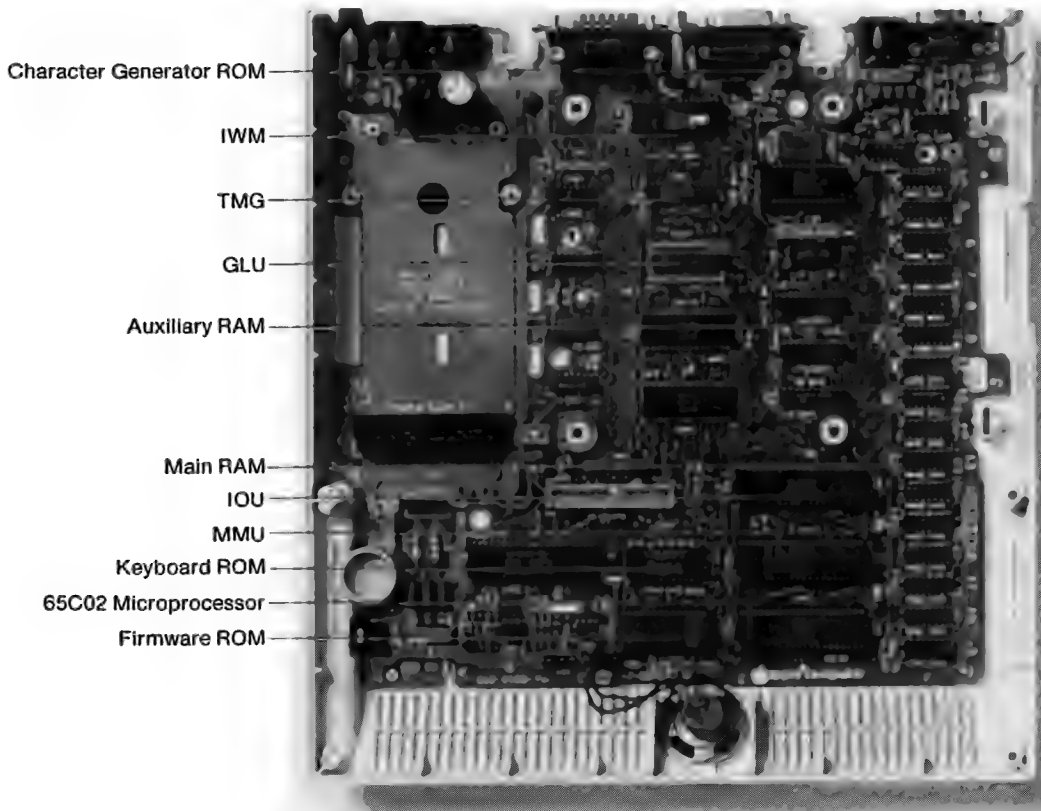
---

## The main logic board

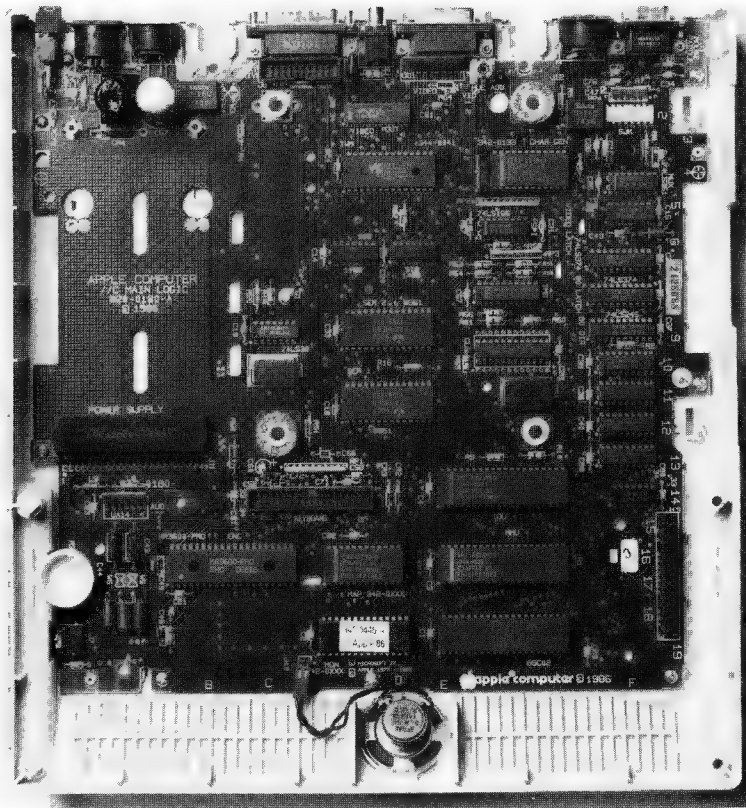
The main logic board, which is mounted flat in the bottom of the Apple IIc's case, has almost all the electronic parts of the computer attached to it.

Figure 1-11 shows the main logic board and the most important integrated circuits (ICs) in the Apple IIc. They are the CPU (central processing unit), RAM (random-access memory), ROM (read-only memory) ICs for keyboard encoding, display character generation, and **firmware**, and the five custom ICs.

The processor is a 65C02 microprocessor. The 65C02 is a CMOS version of the 6502 used in other members of the Apple II family. It is an 8-bit microprocessor with a 16-bit address bus. In the Apple IIc, the 65C02 runs at 1 MHz and performs up to 500,000 8-bit operations per second.



**Figure 1-11**  
Original and UniDisk 3.5 IIc main logic board



**Figure 1-12**  
Memory expansion IIc main logic board

The keyboard is scanned by an IC that generates matrix values for a ROM. The value of the ASCII code supplied by the ROM is latched at a specified memory location and is readable by programs.

The character generator ROM converts ASCII character values to a form that the video display can use.

The Applesoft language interpreter is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual*.

The other ROM contains the Monitor, the Applesoft BASIC interpreter, enhanced video firmware, and other input/output firmware. The firmware that this ROM contains is described throughout this manual.

Five of the large ICs on the main logic board are custom-made for the Apple IIc:

- The *memory management unit* (MMU) contains most of the logic that controls memory addressing in the Apple IIc.
- The *input/output unit* (IOU) contains most of the logic that controls the built-in input and output features of the Apple IIc.
- The *timing generator* (TMG) generates all the system and I/O clock and timing signals from a 14-MHz oscillator.
- The *general logic unit* (GLU) performs the remaining required logic functions.
- The *disk controller unit*, also known as the Integrated Woz Machine (IWM), is a single-chip version of the Apple Disk II controller card. It controls the built-in and external disk drives connected to the Apple IIc.

For more on memory addressing, see Chapter 2.

See Chapters 3 through 9.

Chapter 11 discusses the functions of these integrated circuits in some detail.

---

## The other circuit boards

The Apple IIc contains other circuit boards that serve special purposes: a motor-speed control and read/write logic board for the disk drive, and a matrix board for detecting the position of keys pressed. This manual does not discuss these circuit boards.

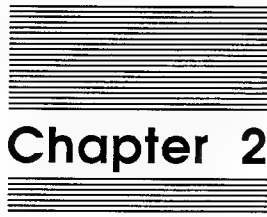
---

### Warning

Adjustment of disk drive speed must be done by an authorized Apple Service Center. Do not attempt to adjust the speed of your built-in disk drive. If you do, you may damage it and you will void your warranty.

---





## **Chapter 2**

# **Memory Organization and Control**

This chapter introduces the Apple IIc's processor, the 65C02, and the memory ranges and locations in the Apple IIc that have been set aside for special purposes. The last section of this chapter describes the reset routines, which restore the computer to a known state.

---

---

## The 65C02 microprocessor

The 65C02 is a general-purpose 8-bit CMOS microprocessor similar in operation to the 6502 used in other members of the Apple II family of computers.

Figure 2-1 is a model of the 65C02 microprocessor's register organization. Registers are fast-acting built-in storage areas where the processor performs and keeps track of its work. The 65C02 has one 16-bit register and five 8-bit registers.

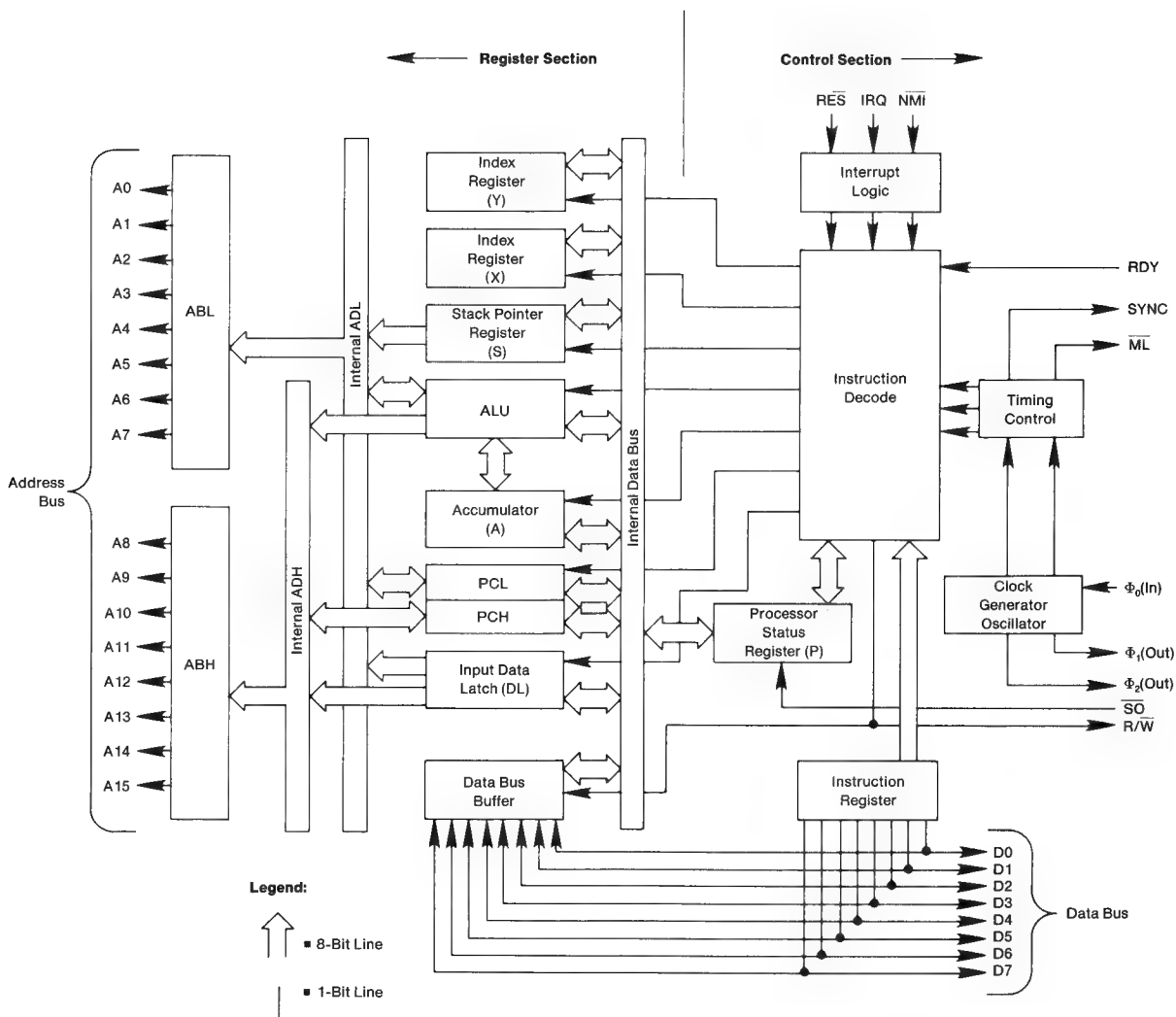
The 16-bit register is called the *program counter* (PC). It specifies the address in memory that contains the instruction the processor is currently carrying out. A 16-bit register can specify any one of 65,536 memory addresses, and so the 65C02 is said to have an address space of 65,536 locations.

The five 8-bit registers in the 65C02 are the following:

- The *accumulator*, or A register. The accumulator is like a desk top where the processor performs mathematical and logical operations on information.
- The *index registers*, X and Y. The processor uses these registers to modify the address where information is to be found or placed, and to pass information from one program to another.
- A *stack pointer*, or S register. The processor uses a 256-byte region of memory—page \$01—as an area to stack up bytes for future use. The stack is empty when the computer is turned on. Several 65C02 instructions either push (store) the contents of a register onto the stack, or pull (retrieve) a byte from the stack and place it in a register. The S register keeps track of the address of the byte in the stack that is currently ready for use.
- A *processor status register*, or P register. Seven of the eight bits of this register are used as flags to record the outcome of processor activities, and can be checked by later instructions to determine what has happened and what the processor should do next.

Each of the other registers holds eight bits (one byte), so the 65C02 is called an *8-bit processor*.

Appendix A lists the instructions the 65C02 can carry out, their use, and their effects on the registers. For further information, consult the pertinent books listed in the Bibliography.



**Figure 2-1**  
 Internal model of the 65C02 microprocessor (copyright © 1982 by NCR Corporation;  
 used by permission)

**Soft switches** are described more fully under “Bank-Switched Memory” and “48K Memory.”

There are two other ROMs in the Apple IIc: one to generate characters corresponding to keystrokes and another to generate characters for display. (See “The Keyboard” and “The Video Display” in Chapter 9.) However, these ROMs are not addressable by the microprocessor.

---

## Overview of the address space

The Apple IIc’s 65C02 microprocessor can address 65,536 (64K) memory locations. All the Apple IIc’s RAM, ROM, and input and output (I/O) devices are accessed using addresses in this 64K address range. Some functions have the same addresses—but not at the same time. The Apple IIc controls its shared addresses by using soft switches. A **soft switch** is a memory location that controls some aspect of the computer’s operation when it is accessed.

All input and output in the Apple IIc is memory mapped—that is, specific memory addresses (all in the \$C0 page) are allocated to each I/O device. In this chapter, the I/O memory spaces are described simply as areas of memory. For details of the built-in I/O features and firmware, refer to the descriptions in Chapters 3 through 9.

A contiguous block of 256 address locations in the 65C02’s address range is called a **page**. A 1-byte address counter or 8-bit register can specify 1 of 256 different locations. Thus, page \$00 consists of memory locations from 0 through 255 (hexadecimal \$00 through \$FF); page \$01 consists of locations 256 through 511 (hexadecimal \$0100 through \$01FF); and so on. In this manual, all page numbers are given in hexadecimal format.

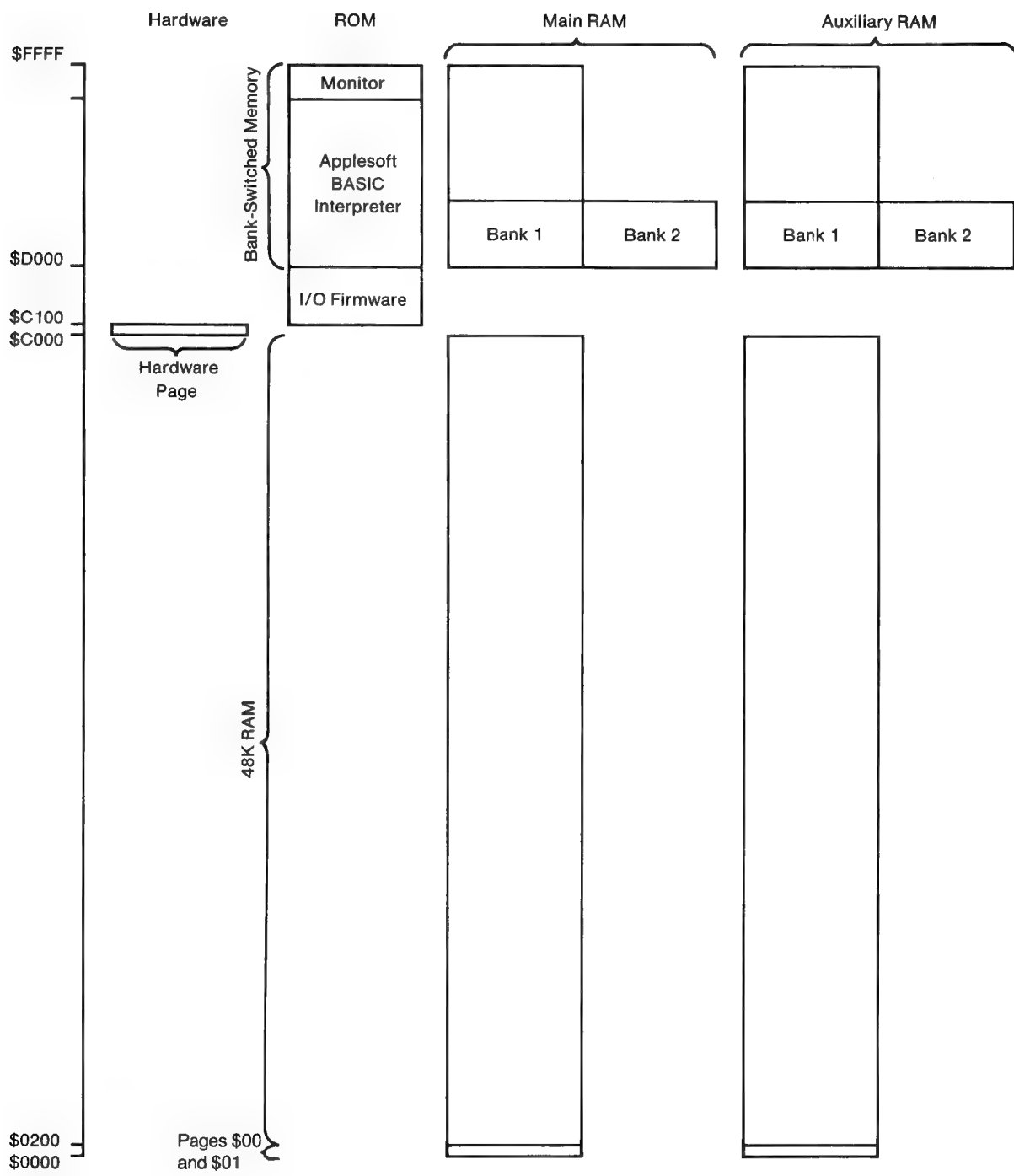
❖ *Note:* The first two digits of a four-digit hexadecimal address are the page number. There are 256 pages of 256 bytes each in the address space. This kind of page is different from the display areas in the Apple IIc, which are sometimes referred to as *Page 1* and *Page 2*. In this manual, dollar signs (\$) in addresses signify that the addresses are in hexadecimal notation.

---

---

## Memory map and memory switching

Figure 2-2 is a map of the Apple IIc’s memory address space and what the major blocks of addresses are used for. As you can see in the figure, addresses \$C000 through \$C0FF contain hardware only, and addresses \$C100 through \$CFFF contain ROM only. At all other addresses there are two to five blocks of RAM or ROM locations. At any given time, only one block of RAM or ROM occupies each set of addresses. As described later in this chapter, soft switches in the hardware page control that blocks the processor is currently using.



**Figure 2-2**  
Apple IIc memory map

---

## Main RAM addresses (\$0000–\$BFFF and \$D000–\$FFFF)

The area labeled *Main RAM* in Figure 2-2 is so called because some or all of it is present in all models of the Apple II series of computers. The Apple IIc has 64K bytes of main RAM.

---

## Auxiliary RAM addresses (\$0000–\$BFFF and \$D000–\$FFFF)

The Apple IIc has 64K of auxiliary RAM built in. Some or all of that range of auxiliary memory is present in an Apple IIe with one of the 80-column text cards installed (see Appendix F), but there is no auxiliary RAM in the Apple II or II Plus.

A range of addresses in auxiliary RAM cannot be used simultaneously with the same range of addresses in main RAM; your programs must use the soft switches described in this chapter to select either main or auxiliary memory for any given range of addresses.

---

## ROM addresses (\$C100–\$FFFF)

ROM addresses contain the built-in Apple IIc firmware. Addresses \$C100 through \$CFFF belong exclusively to ROM. Addresses \$D000 through \$FFFF are shared by ROM, main RAM, and auxiliary RAM; the selection techniques are described later in this chapter.

The Apple IIc's built-in ROM pages \$C1 through CF (addresses \$C100 through \$CFFF) contain I/O firmware. The Apple IIc I/O firmware is roughly divided among the built-in I/O devices as follows:

- Serial port 1 (RS-232 device) firmware entry points are on page \$C1. Much, but not all, of the firmware for the port is in the \$C100 space.
- Serial port 2 (communication device) firmware entry points are on page \$C2. Much, but not all, of the firmware for the port is in the \$C100 space.

- Video output firmware entry points are on page \$C3; the enhanced video firmware and miscellaneous I/O support routines occupy pages \$C8 through \$CF. This is partly because there are no slots 8 through F on the Apple IIc and because the firmware takes up more than one page of firmware memory space.
- Mouse firmware entry points are on page \$C4 (page \$C7 in the memory expansion version).
- Block device I/O firmware entry points are on page \$C6.
- ❖ *Note:* This correspondence of ports and entry points does not imply that all of each port's firmware occupies a specific page. The Apple IIc I/O port firmware space is allocated in a way that provides the best possible performance in the available space.

The operation of the Applesoft interpreter firmware is described in the *Applesoft BASIC Programmer's Reference Manual*.

The ROM address range of pages \$D0 through \$FF contain the Applesoft BASIC interpreter and the Monitor firmware, allocated as follows:

- Pages \$D0 through \$F7 (addresses \$D000 through \$F7FF) contain the Applesoft interpreter firmware.
- Pages \$F8 through \$FF (addresses \$F800 through \$FFFF) contain the Monitor, described in Chapter 10. You can use some of the built-in Monitor routines to make input and output procedures in your assembly-language programs easier to write. These routines are described in Chapters 3 through 9.

---

## Hardware addresses (\$C000–\$C0FF)

The soft switches that the Apple IIc and your programs use to control the Apple IIc's built-in input and output functions are all found in the \$C0 memory page (addresses \$C000 through \$C0FF). In the same range of memory are the switches for selecting blocks of memory throughout the address space. This chapter describes the address space (memory) switches.

The hardware functions of the switches in this page fall into five basic categories:

- *Data inputs.* The only data input is location \$C000, where the low-order seven bits (bits 6 through 0) represent the keyboard key just pressed. (These data are guaranteed valid only when bit 7 = 1.)
- *Flag inputs.* Most built-in input locations are single-bit flags in the high-order (bit 7) position of their respective memory addresses. Flags have only two values: on (greater than or equal to 128 or \$80) or off (less than 128 or \$80).

Chapters 3 through 9 describe the Apple IIc's input and output locations. Appendix B lists these locations in address order, rather than by function.

Bit numbering in a byte is explained in Appendix H.

The switch, hand controller (analog) and button inputs, and the keyboard strobe are examples of flag inputs. The locations for reading soft-switch states are also of this type.

- *Strobe outputs.* The clear keyboard strobe (Chapter 4) and paddle timer strobe (Chapter 9) outputs are controlled by memory locations. If your program reads the contents of one of these locations, then the function associated with that location will be activated.
- *Toggle switches.* The Apple IIc has only one toggle switch: the speaker switch. A toggle switch has only one address assigned to it; each time you access it, it changes to its other state (on or off).

Reading the speaker toggle at location \$C030 clicks the speaker once. However, if you write to the speaker location, the microprocessor activates the address bus twice during successive clock cycles, causing the speaker toggle to end up in its original state before the speaker cone can move. Therefore, you should read, rather than write, to use this device.

The processor cannot read the on/off status of the speaker switch.

- *Soft switches.* Soft switches are two-position switches turned on by accessing one address and turned off by accessing another address. Most of these switches have a third address associated with them for reading the state of the switch.

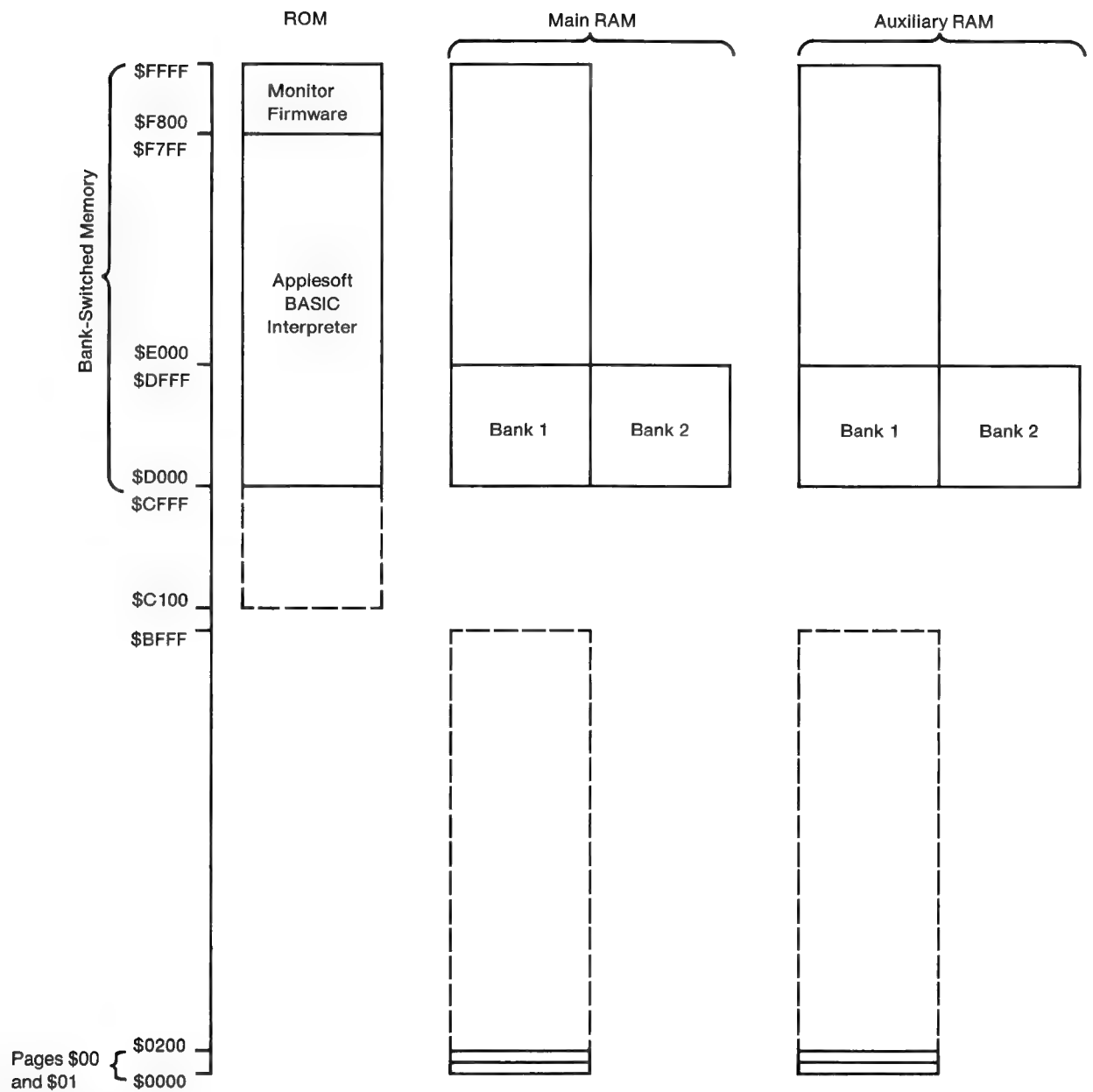
There are eight soft switches that select different combinations of bank-switched memory. Four of these eight switches require that your program read them twice in succession to activate them.

---

---

## Bank-switched memory

The memory areas described in this section are called *bank-switched memory* (Figure 2-3) because so many banks (ranges) of addresses—one bank of ROM and up to four banks of RAM—occupy the same group of locations among the upper addresses of memory. Pages \$00 and \$01, at the low end of memory, are included here because the two sets of them—one in main RAM and one in auxiliary RAM—are controlled by the same switches as the high-address banks. The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K memory space.



**Figure 2-3**  
Bank-switched memory map

---

## Page allocations

Pages \$00 and \$01 are used by many of the 65C02 instructions. The ROM and RAM addresses in bank-switched memory are usually occupied by system software such as interpreters, compilers, and operating systems.

### Page \$00 (one-byte addresses)

Several of the 65C02 microprocessor's addressing modes—for example, indirect addressing—require the use of addresses in page \$00, or zero page. However, the Monitor, the interpreters, and the operating systems all make extensive use of page \$00, too. One way to avoid conflicts is to use only those page-\$00 locations not already used by these other programs. But there is another way.

As you can see from Table B-1 in Appendix B, page \$00 is pretty well used up, except for a few bytes here and there. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: turn off interrupts, save the contents of part of page \$00, use that part, then restore the previous contents to page \$00, restore interrupts to their previous state, and then pass control to another program.

### Page \$01 (the 65C02 stack)

The 65C02 microprocessor uses page \$01 as its stack—a place where it can store subroutine return addresses, in last-in, first-out sequence. Programs can also use the stack for temporary storage of registers (via push and pull instructions). However, programs should use the stack carefully.

### Pages \$D0–\$FF (ROM and RAM)

These memory banks are controlled by the soft switches described under "Using Bank Selector Switches."

The memory address space from locations \$D000 through \$FFFF is used for both ROM and RAM. The 12K bytes of ROM in this address space contain the Monitor and the Applesoft BASIC interpreter.

There are 16K bytes of main RAM in this 12K space, with two banks occupying the 4K of addresses from \$D000 through \$DFFF. The RAM is normally used for storing other languages such as Pascal, or operating systems such as ProDOS®.

There are also 16K bytes of auxiliary RAM in this 12K space, again with double occupancy in the address range \$D000 through \$DFFF.

---

## Using bank selector switches

You switch banks of memory in the same way you switch other functions in the Apple IIc: by using soft switches. These soft switches do four things:

- ☐ select either RAM or ROM in this memory space
- ☐ allow or inhibit (write-protect) writing to the RAM when RAM is selected
- ☐ select the first or second 4K-byte bank of RAM in the address space \$D000 through \$DFFF
- ☐ select either main RAM or auxiliary RAM

---

**Warning** Do not use soft switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program.

---

Table 2-1 shows the addresses of the soft switches for selecting all allowed combinations of reading and writing in this memory space, and the addresses of the locations to read the switch settings. Figures 2-4 through 2-10 illustrate how to select the combinations and what the resulting status of each switch is.

To make sure you do not inadvertently remove write protection from bank-switched RAM, the four write-enable addresses require that you read them twice in succession (indicated by *RR* in Table 2-1).

Because the AltZP switch shares the read keyboard address, you must write (*W* in Table 2-1) to its locations to change the switch setting.

To find out which way a switch is set, read the appropriate location and then check bit 7 (shown as *R7* in Table 2-1). If the bit is a 1, the answer to the question given in the table is affirmative.

Note that there is no way to check whether write protection is on or off.

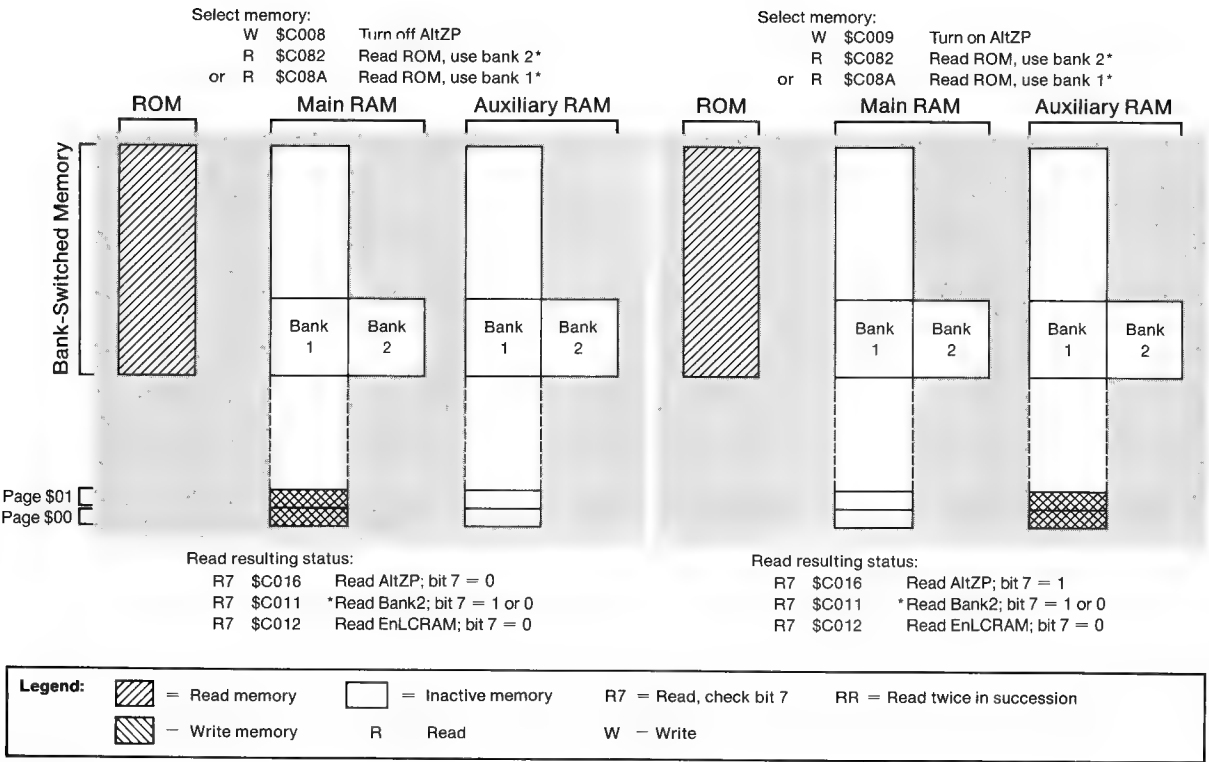
---

**Important** You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well. However, you can read ROM and write RAM (Figures 2-5 and 2-6), which makes it easy to transfer firmware to bank-switched RAM if you want to use it with a program there.

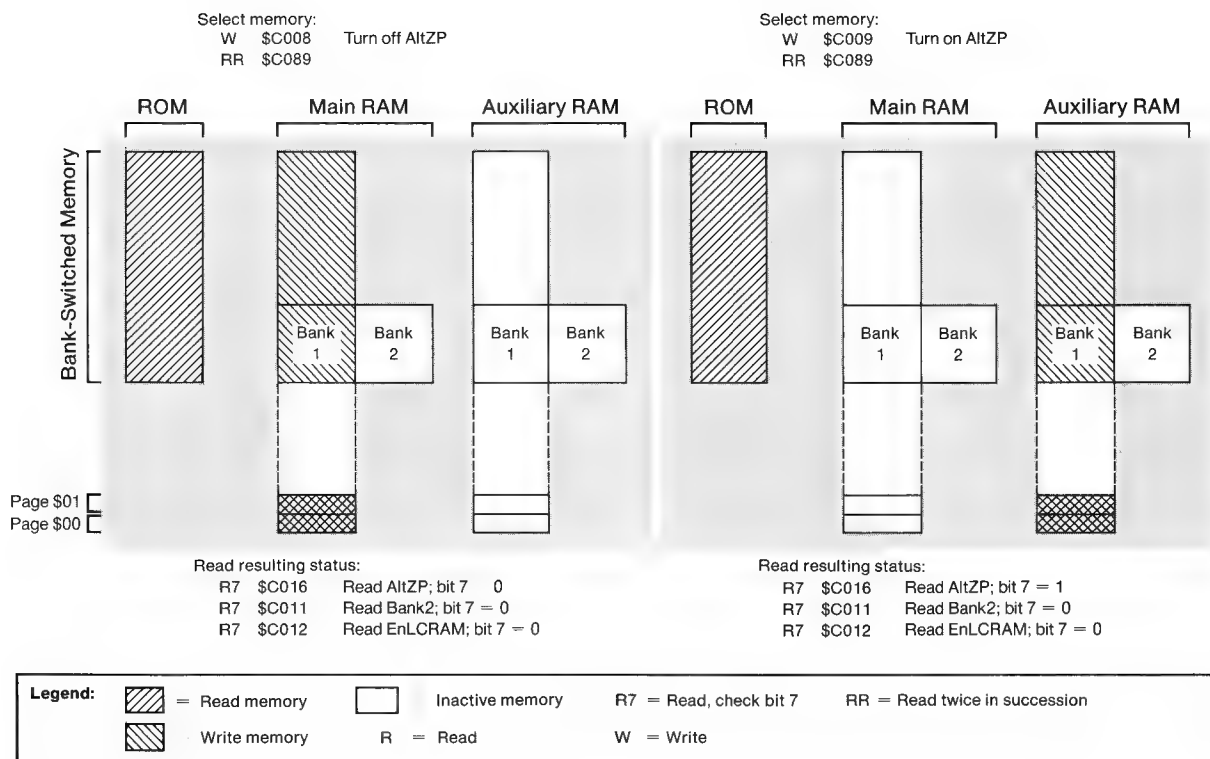
---

**Table 2-1**  
Bank selector switches

Name	Action	Hex	Dec	Function
	R	\$C080	49280	Read RAM; no write; use \$D000 bank 2
	RR	\$C081	49281	Read ROM; write RAM; use \$D000 bank 2
	R	\$C082	49282	Read ROM; no write; use \$D000 bank 2
	RR	\$C083	49283	Read and write RAM; use \$D000 bank 2
	R	\$C088	49288	Read RAM; no write; use \$D000 bank 1
	RR	\$C089	49289	Read ROM; write RAM; use \$D000 bank 1
	R	\$C08A	49290	Read ROM; no write; use \$D000 bank 1
	RR	\$C08B	49291	Read and write RAM; use \$D000 bank 1
RdBnk2	R7	\$C011	49169	Read whether \$D000 bank 2 (1) or bank 1 (0)
RdLCRAM	R7	\$C012	49170	Read RAM (1) or ROM (0)
AltZP	W	\$C008	49160	Off: Use main bank, page \$00 and page \$01
AltZP	W	\$C009	49161	On: Use auxiliary bank, page \$00 and page \$01
RdAltZP	R7	\$C016	49174	Read whether auxiliary (1) or main (0) bank



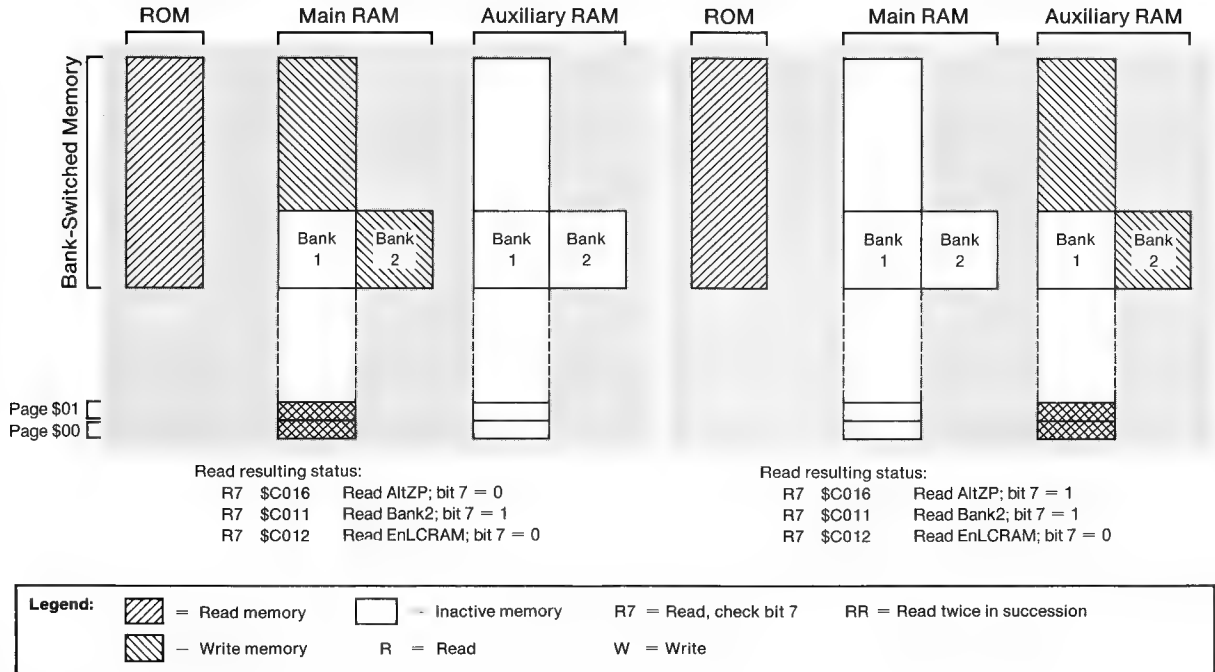
**Figure 2-4**  
Read ROM



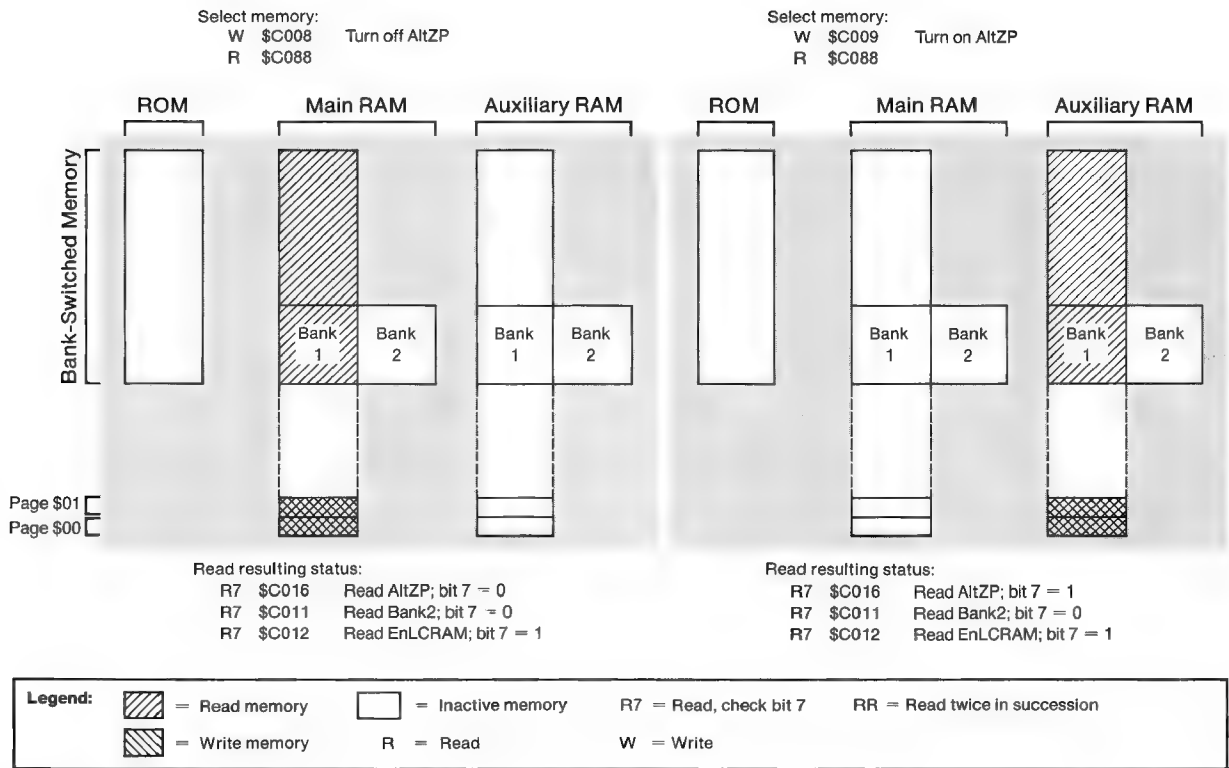
**Figure 2-5**  
Read ROM, write RAM, and use first \$D0 bank

Select memory:  
W \$C008 Turn off AltZP  
RR \$C081

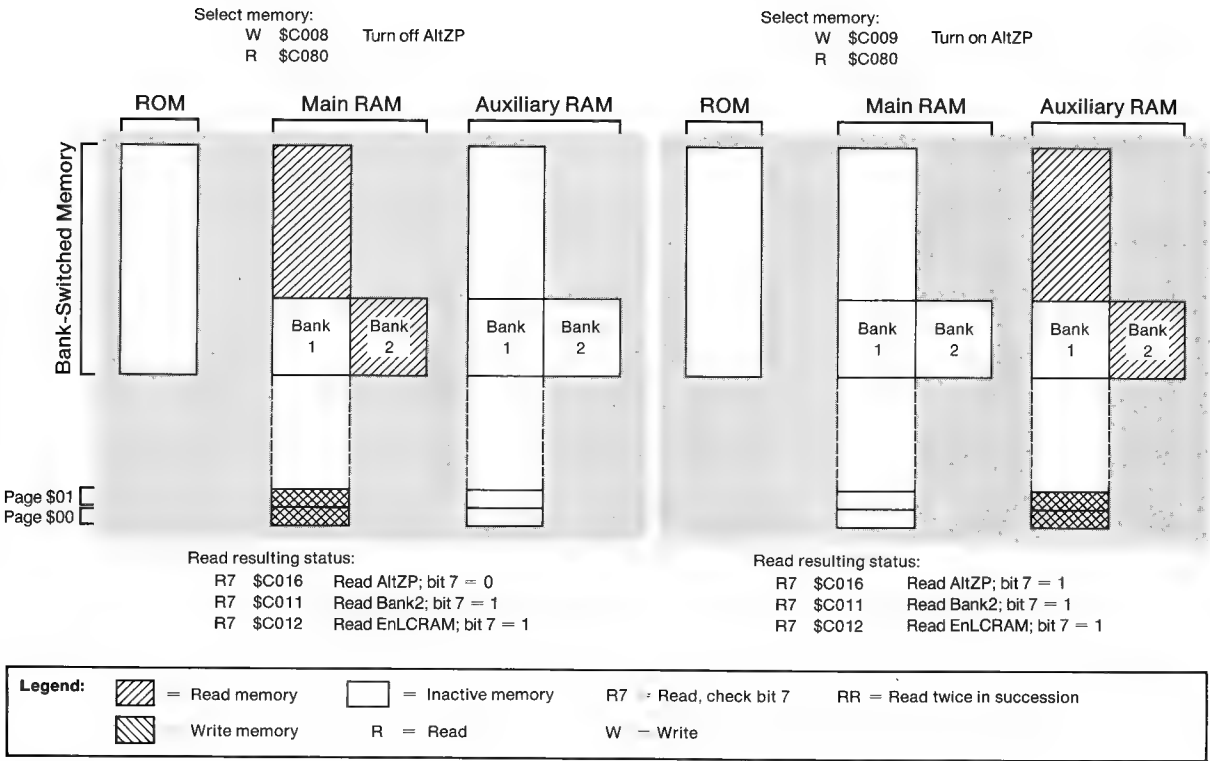
Select memory:  
W \$C009 Turn on AltZP  
RR \$C081



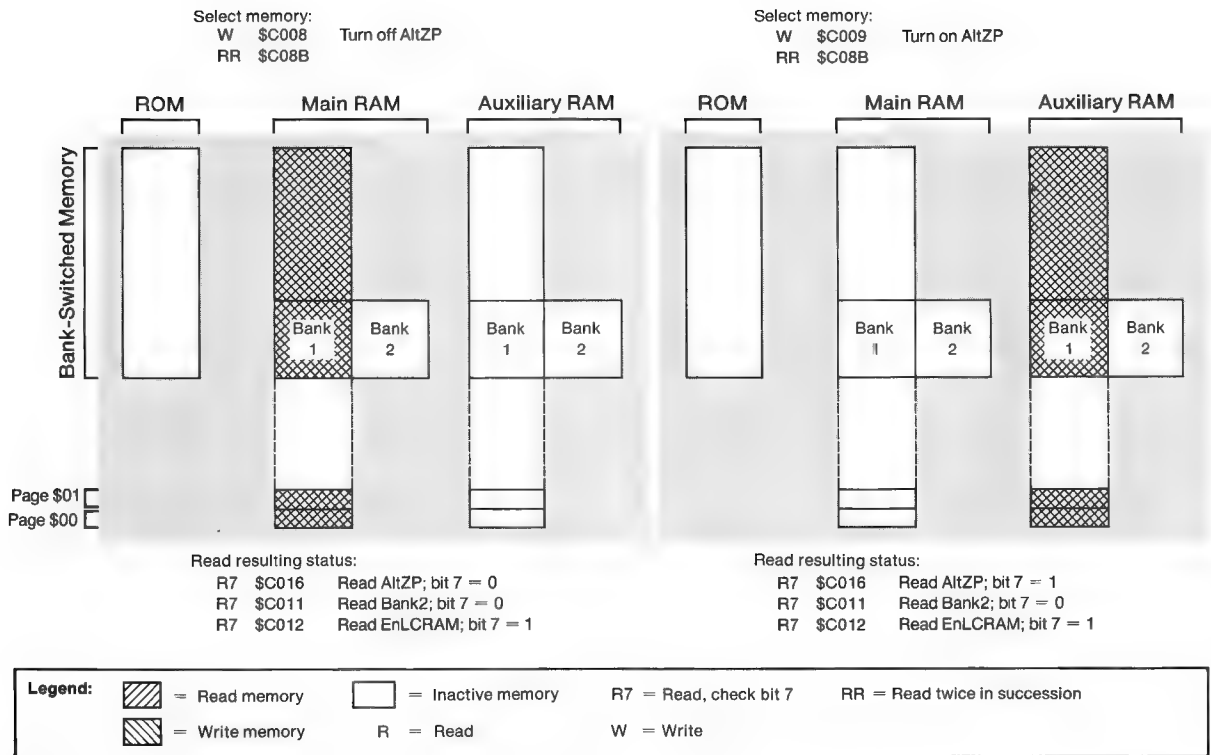
**Figure 2-6**  
Read ROM, write RAM, and use second \$D0 bank



**Figure 2-7**  
Read RAM and use first \$D0 bank



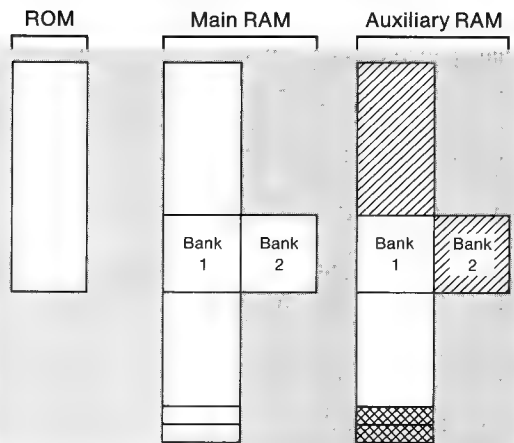
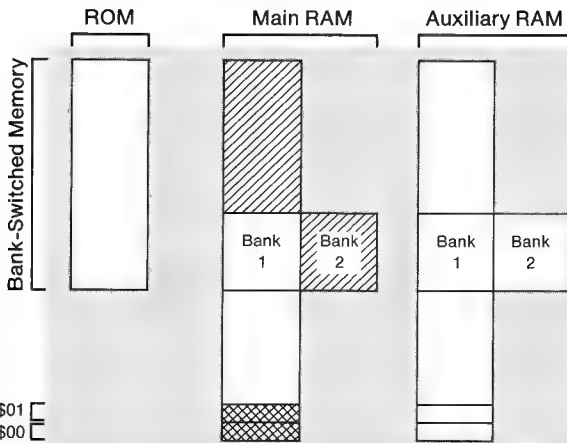
**Figure 2-8**  
Read RAM and use second \$D0 bank



**Figure 2-9**  
Read and write RAM and use first \$D0 bank

Select memory:  
W \$C008 Turn off AltZP  
RR \$C083

Select memory:  
W \$C009 Turn on AltZP  
RR \$C083




Read resulting status:

R7 \$C016 Read AltZP; bit 7 = 0  
R7 \$C011 Read Bank2; bit 7 = 1  
R7 \$C012 Read EnLCRAM; bit 7 = 1

Read resulting status:

R7 \$C016 Read AltZP; bit 7 = 1  
R7 \$C011 Read Bank2; bit 7 = 0  
R7 \$C012 Read EnLCRAM; bit 7 = 1

**Legend:**

 = Read memory	 = Inactive memory	R7 = Read, check bit 7	RR = Read twice in succession
 = Write memory	R = Read	W = Write	

**Figure 2-10**  
Read and write RAM and use second \$D0 bank

---

---

## 48K memory

The 48K memory space (actually, 47.5K) extends from location \$0200 to location \$BFFF (Figure 2-11) in both main and auxiliary RAM. The amount of storage available in this address space depends on what language or operating system you are using, and what video display needs your program has.

---

### Page allocations

Most of the Apple IIc's 48K RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware, the Applesoft BASIC interpreter, and whatever video display you may select.

---

#### Important

The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain.

---

A **buffer** is any storage area set aside for one program or device to put information into and another to take information out of at a different time or rate.

#### Page \$02 (the input buffer)

The GetLn input routine uses page \$02 as its keyboard-input **buffer**. The size of this buffer (256 bytes) sets the maximum size of input strings read by Applesoft or the Monitor. If you know that you won't be typing any long input strings (more than, say, 30 characters), you can store temporary data at the upper end of page \$02.

Refer to Appendix D and to the appropriate programmer and reference manuals for operating system use of page \$03.

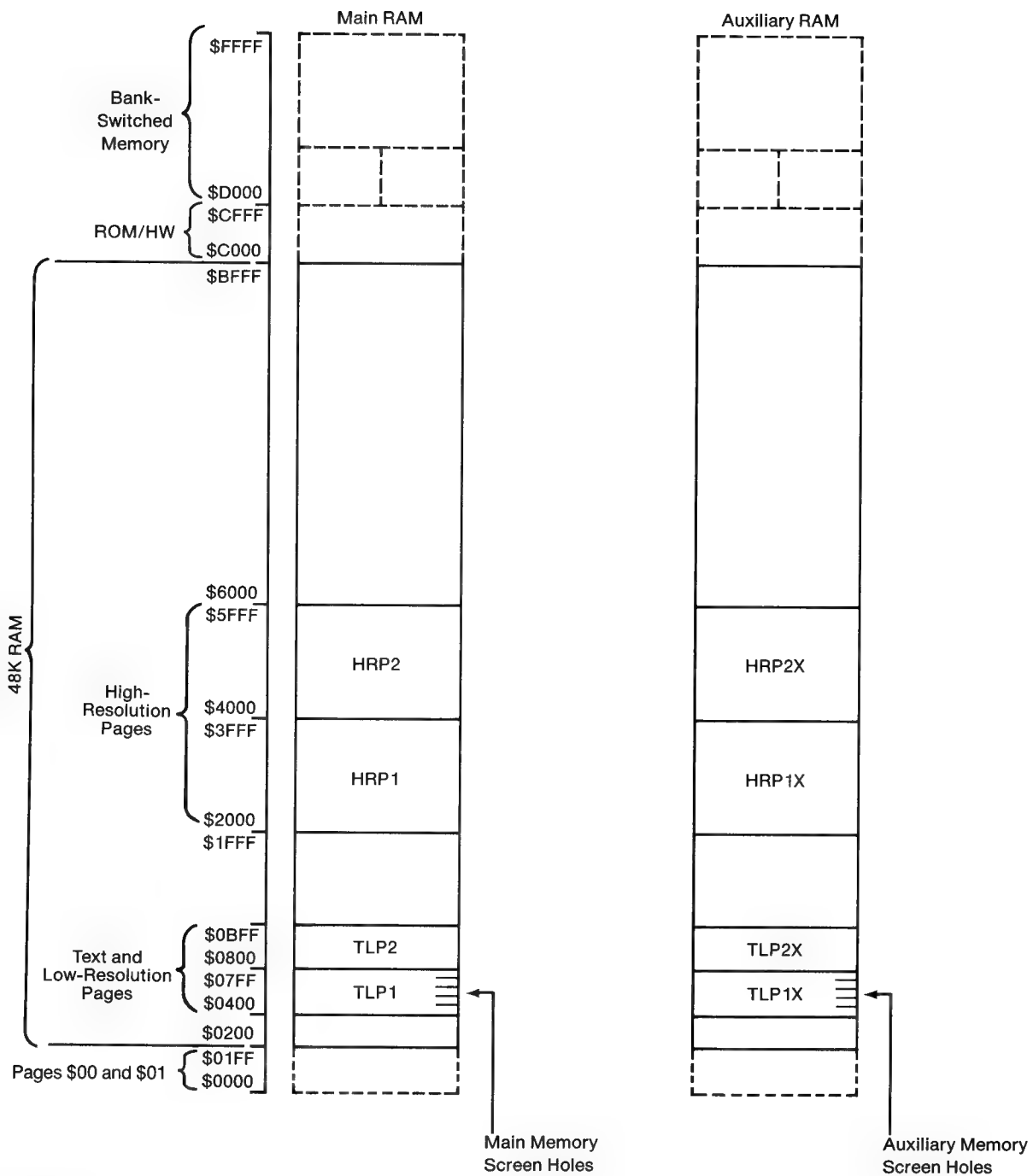
#### Page \$03 (global storage and vectors)

The Monitor and operating systems use parts of page \$03 for **global storage** and **vectors**. Table 2-7, later in this chapter, shows the part of page \$03 the built-in firmware uses.

**Global storage** refers to an area reserved for information that programs use in common. **Vectors**—the addresses of special routines—are examples of this kind of information. See "The Reset Routine" about the global storage and vectors found on page \$03.

#### Pages \$04–\$07 (text and low-resolution Page 1)

The most often used display buffer is the text and low-resolution graphics Page 1 (TLP1 in Figure 2-11), which occupies main memory pages \$04 through \$07. It is not usable for program and data storage if you are using Monitor routines or Applesoft, or with almost any other program that uses text or low-resolution display.



**Figure 2-11**  
48K memory map

Text and low-resolution Page 1X (TLP1X) is an identical display page occupying auxiliary memory pages \$04 through \$07. This pair of text and low-resolution graphics pages are used together to produce 80-column text display.

See “Port Screen Hole RAM Space” in Chapter 3.

There are 128 locations in pages \$04 through \$07 (64 in main RAM, 64 in auxiliary RAM) that are not displayed on the screen. These locations are called *screen holes*.

---

**Warning** The screen holes are reserved for use by the built-in firmware.

---

### **Pages \$08–\$0B (text and low-resolution Page 2)**

The second text and low-resolution graphics display buffer, TLP2, occupies main memory pages \$08 through \$0B. Most programs do not use Page 2 for displays, but TLP2 is there for display use if required.

Text and low-resolution Page 2X (TLP2X) is an identical display buffer occupying pages \$08 through \$0B in auxiliary memory.

Note that Apple IIc firmware does not provide a way to use the second pair of text and low-resolution graphics pages for 80-column text display.

### **Page \$08 (communication port buffers)**

For more on serial port 2, see Chapter 8.

Serial port 2 uses the first half of auxiliary memory page \$08 (addresses \$0800 through \$087F) as a keyboard input buffer, and the second half of the page (addresses \$0880 through \$08FF) as a serial input buffer. These buffers increase the data transfer rates possible with the serial communication port. Appendix E explains how to use these features. If your program does not use this page for buffers, it can use it as part of TLP2X.

### **Pages \$20–\$3F (high-resolution Page 1)**

The primary high-resolution graphics display buffer, high-resolution Page 1 (HRP1), occupies the 32 memory pages from \$20 through \$3F (locations \$2000 through \$3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-resolution Page 1X (HRP1X) is an identical display page occupying auxiliary memory pages \$20 through \$3F.

See Chapter 5.

The Apple IIc can display double high-resolution graphics by interleaving HRP1 and HRP1X.

## Pages \$40–\$5F (high-resolution Page 2)

High-resolution Page 2 occupies main memory pages \$40 through \$5F (locations \$4000 through \$5FFF). Most programs use this area for program or data storage, but it is also available as a second high-resolution page.

High-resolution Page 2X (HRP2X) occupies auxiliary memory pages \$40 through \$5F.

Apple IIc firmware provides high-resolution graphics routines for HRP1 and HRP2 only. Refer to the *Applesoft BASIC Programmer's Reference Manual*.

---

## Using 48K memory switches

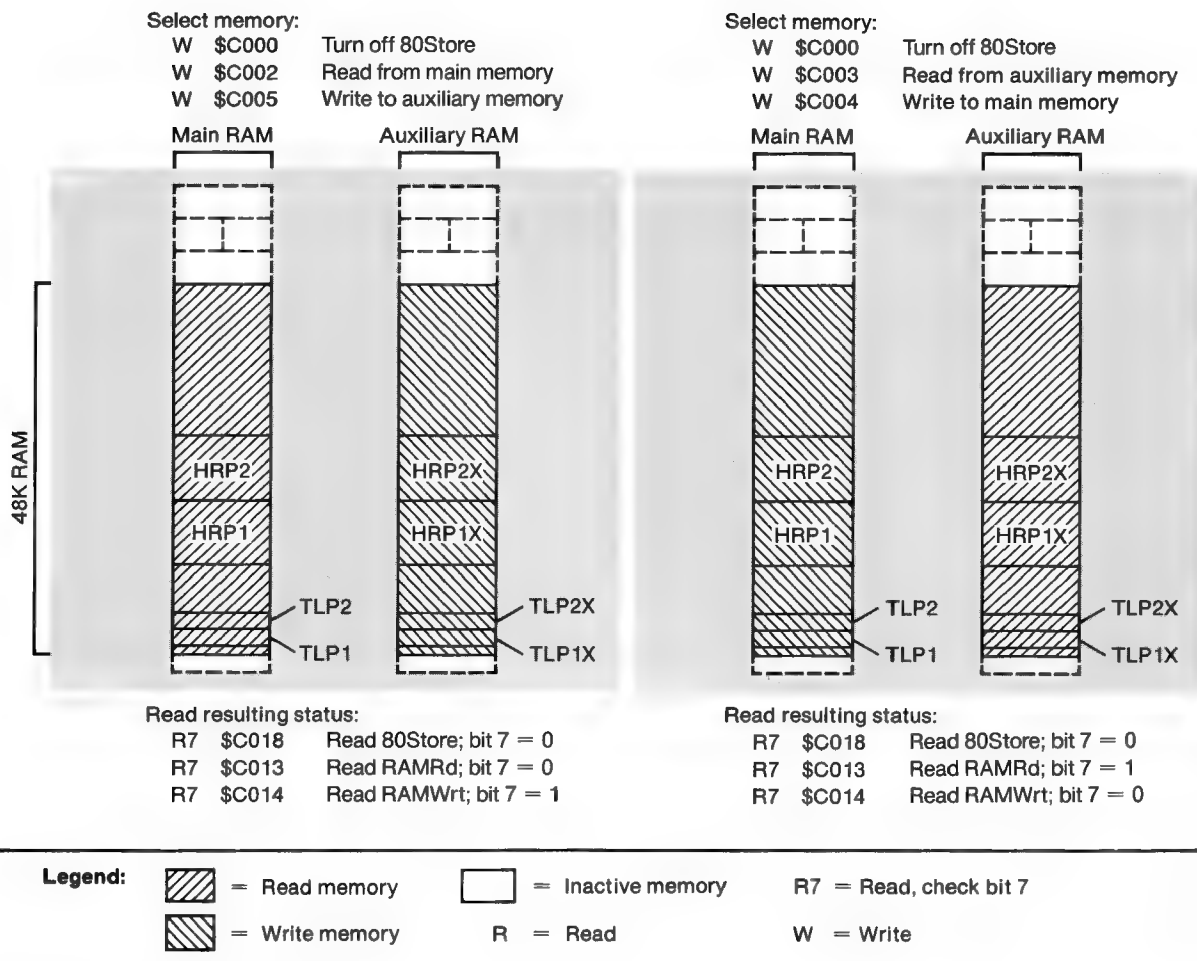
Two switches select main or auxiliary RAM in the 48K memory space: RAMRd determines which to use for reading, and RAMWrt determines which to use for writing. When these switches are on, they select auxiliary memory. When they are off, they select main memory. (This discussion assumes that the 80Store switch, used to control display memory, is off.)

Each switch has three locations assigned to it (Table 2-2): one to turn it on, one to turn it off, and a third to read its state. Because the memory locations for turning the switches on and off are shared with keyboard reading functions, you must write to these addresses to use them for memory switching. For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

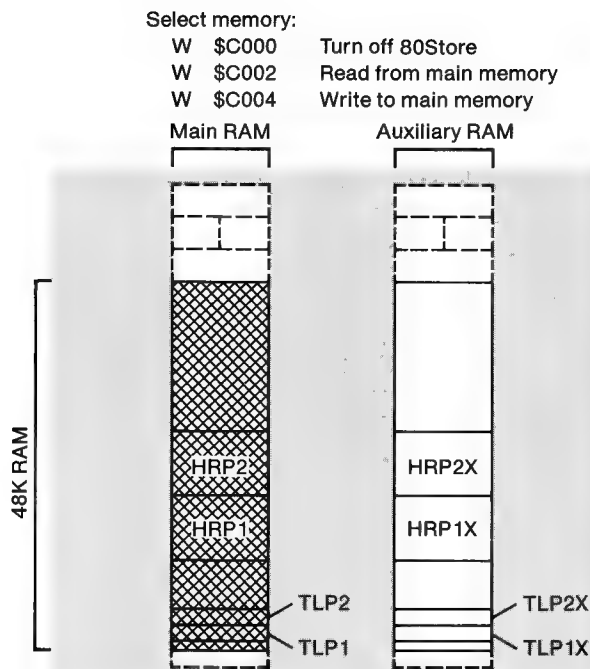
**Table 2-2**  
48K memory switches

Name	Action	Hex	Dec	Function
RAMRd	W	\$C002	49154	Off: Read main 48K RAM
RAMRd	W	\$C003	49155	On: Read auxiliary 48K RAM
RdRAMRd	R7	\$C013	49171	Read whether main (0) or aux. (1)
RAMWrt	W	\$C004	49156	Off: Write to main 48K RAM
RAMWrt	W	\$C005	49157	On: Write to auxiliary 48K RAM
RdRAMWrt	R7	\$C014	49172	Read whether main (0) or aux. (1)

*Note:* 80Store must be off to switch all memory in this range, including display memory (Table 2-6).



**Figure 2-12**  
48K RAM selection, split pairs

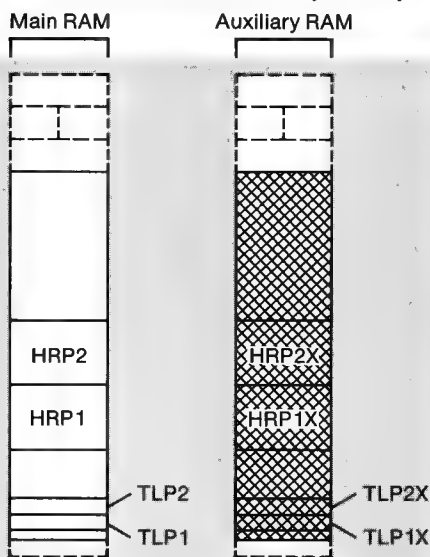


Read resulting status:

R7 \$C018	Read 80Store; bit 7 = 0
R7 \$C013	Read RAMRd; bit 7 = 0
R7 \$C014	Read RAMWrt; bit 7 = 0

Select memory:

W \$C000	Turn off 80Store
W \$C003	Read from auxiliary memory
W \$C005	Write to auxiliary memory



Read resulting status:

R7 \$C018	Read 80Store; bit 7 = 0
R7 \$C013	Read RAMRd; bit 7 = 1
R7 \$C014	Read RAMWrt; bit 7 = 1

**Legend:**



= Read memory

= Write memory



= Inactive memory

R = Read

R7 = Read, check bit 7

W = Write

**Figure 2-13**

48K RAM selection, one side only

---

## Transfers between main and auxiliary memory

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in 48K RAM transfer routines. These routines (listed in Table 2-3) make it possible to move between main and auxiliary memory without having to manipulate the soft switches described earlier in this chapter.

---

**Important** The routines described below make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid catastrophic effects on your program.

---

**Table 2-3**  
48K RAM transfer routines

Name	Action	Hex	Function
MoveAux	JSR	\$C311	Move data blocks between main and auxiliary 48K memory.
XFer	JMP	\$C314	Transfer program control between main and auxiliary 48K memory.

### Transferring data

In your assembly-language programs, you can use the built-in routine named MoveAux to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page \$00 and set or clear the carry bit to select the direction of the move.

---

**Warning** Don't try to use MoveAux to copy data in bank-switched memory (page \$00, page \$01, or pages \$D0 through \$FF). MoveAux uses page \$00 all during the copy.

---

The pairs of bytes you use for passing addresses to this routine are called A1, A2, and A4, and they are used for parameter passing by several of the Apple IIc's built-in routines. The addresses of these byte pairs are shown in Table 2-4.

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

**Table 2-4**  
Parameters for MoveAux routine

Name	Location	Parameter passed
Carry		1 = Move from main to auxiliary memory. 0 = Move from auxiliary to main memory.
A1L	\$3C	Source starting address, low-order byte.
A1H	\$3D	Source starting address, high-order byte.
A2L	\$3E	Source ending address, low-order byte.
A2H	\$3F	Source ending address, high-order byte.
A4L	\$42	Destination starting address, low-order byte.
A4H	\$43	Destination starting address, high-order byte.
	X, Y, A	These registers are preserved.

The MoveAux routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit (SEC instruction); to copy data from auxiliary memory to main memory, clear the carry bit (CLC instruction).

When you make the subroutine call to MoveAux, the subroutine copies the block of data as specified by the A register and the carry bit. When it is finished, the accumulator and the X and Y registers are just as they were when you called it.

## Transferring control

You can use the built-in routine named *XFer* to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFer: the address of the routine you are transferring to, the direction of the transfer, and which page \$00 and stack you want to use (Table 2-5).

**Table 2-5**  
Parameters for XFer routine

Name	Location	Parameter passed
Carry		1 = Transfer from main to auxiliary memory. 0 = Transfer from auxiliary to main memory.
Overflow		1 = Use page \$00 and stack in auxiliary memory. 0 = Use page \$00 and stack in main memory.
	\$03ED	Program starting address, low-order byte.
	\$03EE	Program starting address, high-order byte.
	X, Y, A	These registers are preserved.

Put the transfer address into the two bytes at locations \$03ED and \$03EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory.

Use the overflow bit to select which page \$00 and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit (cause an overflow condition) to use the auxiliary memory.

After you have set up the parameters, pass control to the XFer routine by a jump instruction, rather than a subroutine call.

---

**Warning**

It is your responsibility as the programmer to save the current stack pointer before using XFer and to restore it after regaining control. Failure to do so will cause program errors. Refer to Appendix E for instructions on how to do this.

---

---

## Using display memory switches

Selection of main or auxiliary RAM for the 48K memory space is described earlier in this chapter. However, under many circumstances your program may want to control reading and writing to display pages separately. The switches discussed in this section override the effects of RAMRd and RAMWrt for display pages only.

Three switches are involved in the display page selection process. Each of them has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state (Table 2-6). One of the switches, 80Store, shares its on and off addresses with a keyboard reading function. As a result, your program must write to these locations to turn the switch on and off.

**Table 2-6**  
Display memory switches

Name	Action	Hex	Dec	Function
80Store	W	\$C000	49152	Off: RAMRd and RAMWrt determine RAM locations.
80Store	W	\$C001	49153	On: Page2 switches between TLP1 and TLP1X, and (if HiRes on) between HRP1 and HRP1X.
Rd80Store	R7	\$C018	49176	Read whether 80Store on (1) or off (0).
Page2	R	\$C054	49236	Off: Select TLP1 and HRP1.
Page2	R	\$C055	49237	On: If 80Store off, switch to TLP2, and (if HiRes on) to HRP2. If 80Store on, switch to TLP1X, and (if HiRes on) to HRP1X.
RdPage2	R7	\$C01C	49180	Read whether Page2 on (1) or off (0).
HiRes	R	\$C056	49238	Off: Display text and low-resolution page.
HiRes	R	\$C057	49239	On: Display high-resolution pages; make Page2 switch between high-resolution pages.
RdHiRes	R7	\$C01D	49181	Read whether HiRes on (1) or off (0).
IOUDis	W	\$C07E	49278	On: Disable IOU access for addresses \$C058 to \$C05F; enable access to DHiRes switch*.
IOUDis	W	\$C07F	49279	Off: Enable IOU access for addresses \$C058 to \$C05F; disable access to DHiRes switch*.

**Table 2-6 (continued)**  
Display memory switches

Name	Action	Hex	Dec	Function
RdIOUDis	R7	\$CO7E	49278	Read IOUDis switch (1=off) <sup>†</sup> .
DHiRes	R/W	\$CO5E	49246	On: (If IOUDis on) turn on double high-resolution.
DHiRes	R/W	\$CO5F	49247	Off: (If IOUDis on) turn off double high-resolution.
RdDHiRes	R7	\$CO7F	49279	Read DHiRes switch (1=on) <sup>†</sup> .

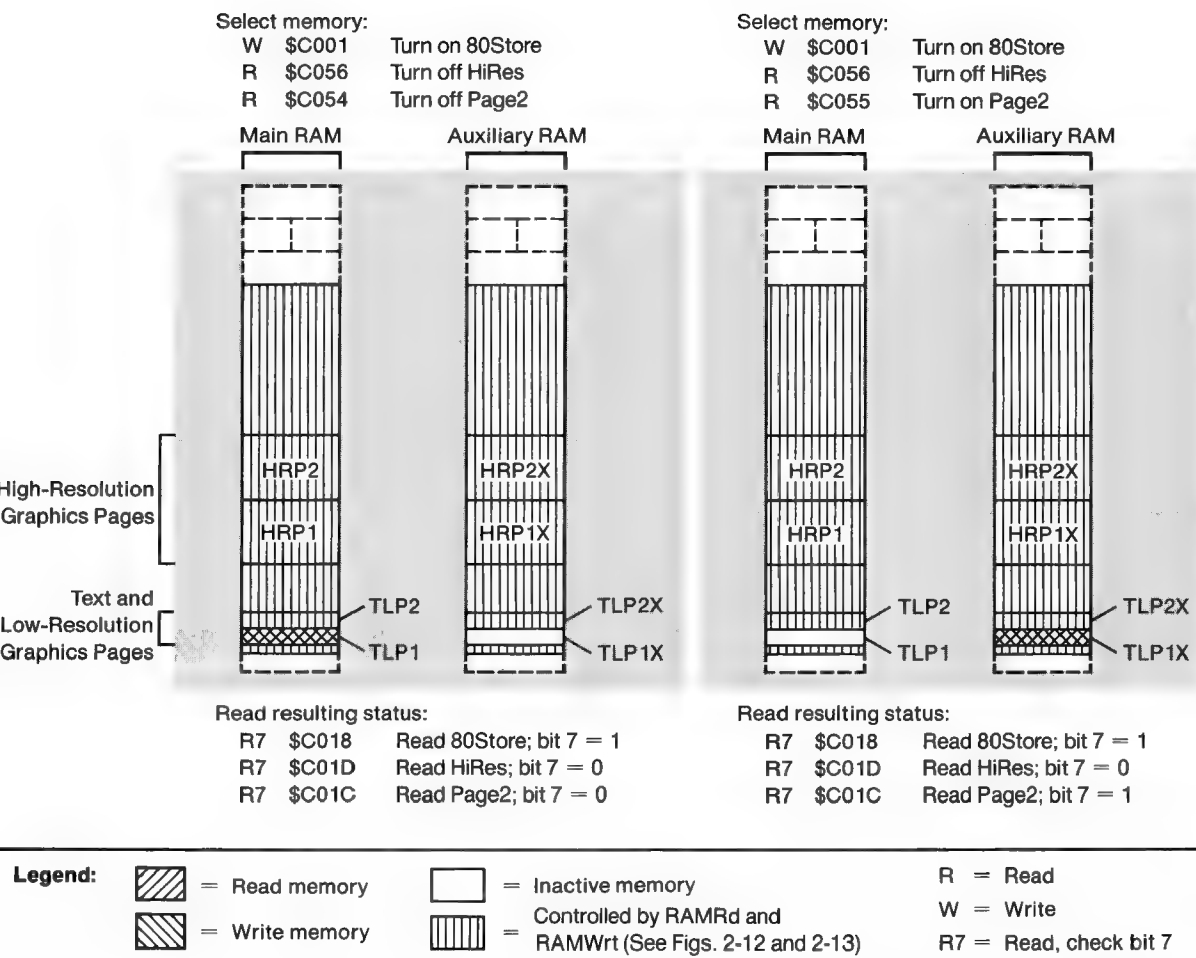
\* The firmware normally leaves IOUDis on.

† Reading or writing any address in the range \$CO70–\$CO7F also triggers the paddle timer and resets VBLInt (see Chapter 9).

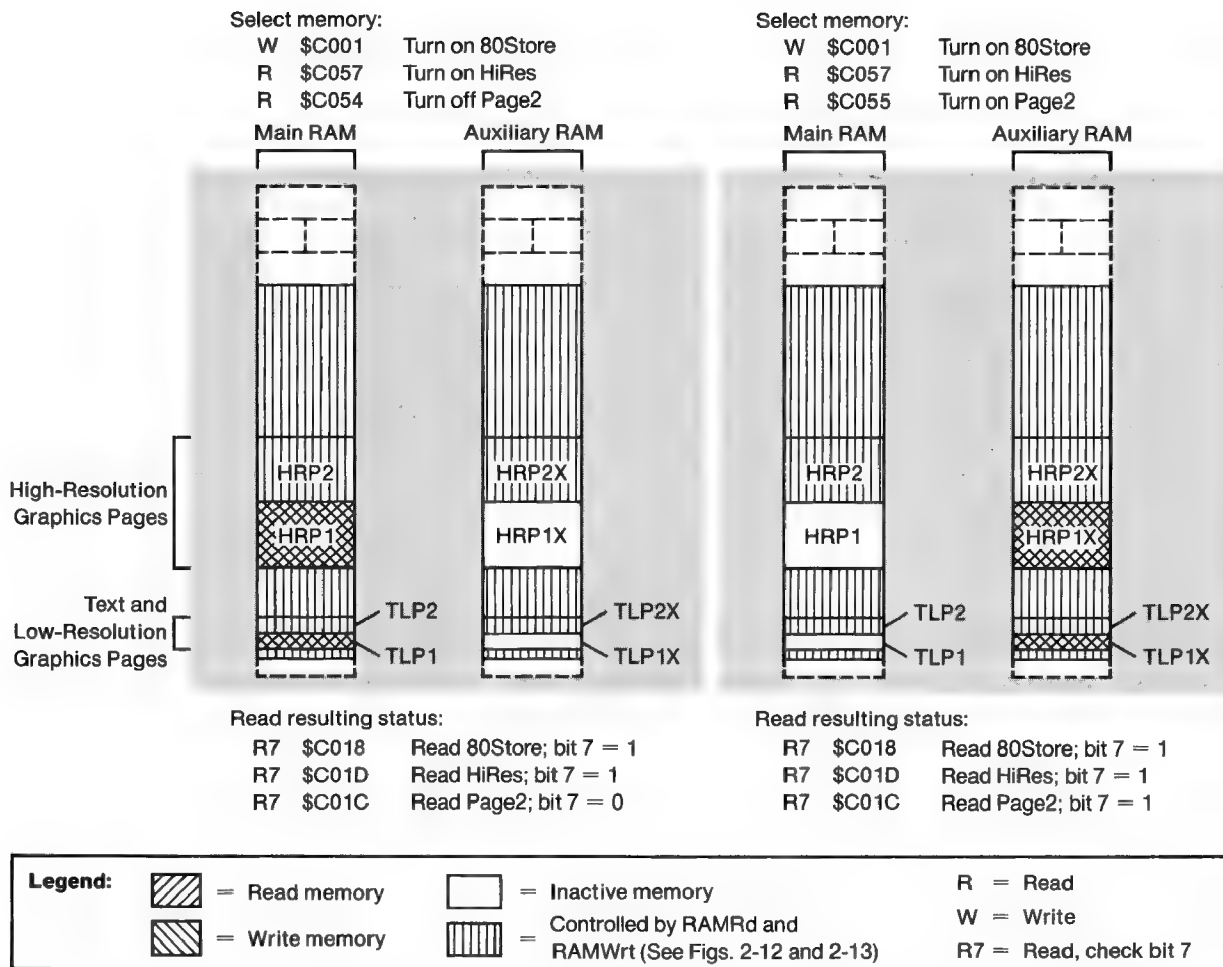
For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

Here is how these switches work for reading and writing:

- If HiRes is off, then Page2 switches between text and low-resolution graphics pages (TLP) only. If HiRes is on, then Page2 switches between TLP and high-resolution graphics pages (HRP).
- If 80Store is off, RAMRd and RAMWrt (Table 2-2) determine whether main or auxiliary RAM locations are used. Page2 selects pages for display (Chapter 5), but not for reading and writing.
- If 80Store is on, it overrides RAMRd and RAMWrt with respect to the display pages selected by HiRes and Page2 (Figures 2-14 and 2-15).



**Figure 2-14**  
Page2 selections, 80Store on and HiRes off

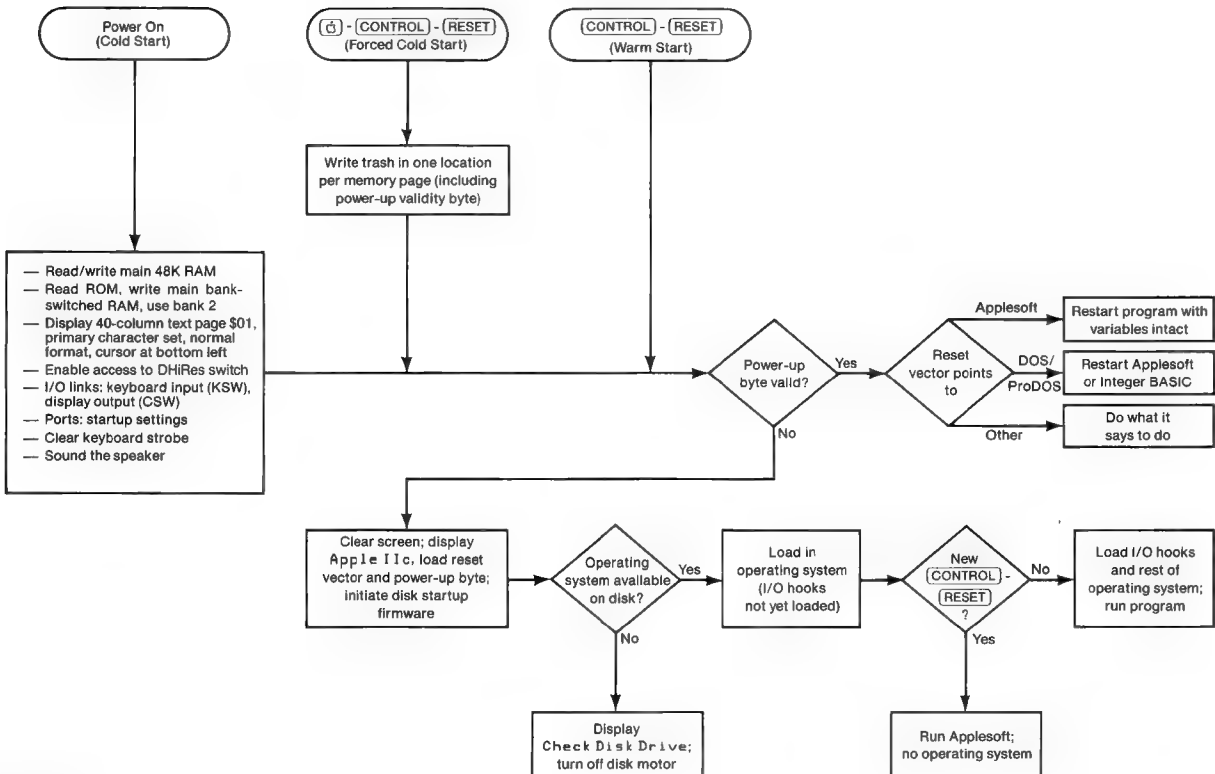


**Figure 2-15**  
Page2 selections, 80Store on and HiRes on

## The reset routine

A procedure called the *reset routine* (Figure 2-16) puts the Apple IIc into a known state when it has just been turned on or when you hold down Control while pressing Reset. The reset routine puts the Apple IIc into its normal operating mode and restarts the program indicated at locations \$03F2 and \$03F3 (Table 2-7).

When you initiate a reset, hardware in the Apple IIc sets the memory-controlling soft switches to normal; main ROM and RAM are enabled, auxiliary RAM is disabled and the bank-switched memory space is set up to read from ROM and write to RAM, using the second bank at \$D000.



**Figure 2-16**  
Reset routine flowchart

**Table 2-7**  
Page \$03 vectors

Vector address	Vector function
\$03F0 (1008)	Address of the subroutine that handles BRK requests (normally \$59, \$FA)
\$03F1 (1009)	
\$03F2 (1010)	Reset vector (see text) \$03F3 (1011)
\$03F4 (1012)	Power-up byte (see text)
\$03F5 (1013)	Jump instruction to the subroutine that handles Applesoft and commands (normally \$4C, \$58, \$FF)
\$03F6 (1014)	
\$03F7 (1015)	
\$03F8 (1016)	Jump instruction to the subroutine that handles user Control-Y commands
\$03F9 (1017)	
\$03FA (1018)	
\$03FB (1019)	Jump instruction to the subroutine that handles nonmaskable interrupts (not used on Apple IIc)
\$03FC (1020)	
\$03FD (1021)	
\$03FE (1022)	Interrupt vector (address of the subroutine that handles interrupt requests) (Appendix E)
\$03FF (1023)	

The reset routine sets the display-controlling soft switches to display 40-column text Page 1 using the primary character set, then sets the display window equal to the full 40-column display, puts the cursor at the bottom of the screen, and sets the text display format to normal.

The reset routine also sets the keyboard and display as the standard input and output devices (Chapter 3). It masks mouse interrupts and sets mouse defaults (Table 9-1). Finally, it enables DHires switch access (by turning on IOUDis), clears the keyboard strobe, and sounds the speaker.

The Apple IIc has three types of reset: *power-on reset*, also called cold-start reset; *warm-start reset*; and *forced cold-start reset*. The procedure described above is the same for any type of reset. What happens next depends on the reset vector. The reset routine checks the reset vector to determine whether it is valid or not. If the reset was caused by turning the power on, the vector will not be valid, and the reset routine will perform the cold-start procedure. If the vector is valid, the routine will perform the warm-start procedure.

The reset vector validity check is described under "The Reset Vector."

---

## The cold-start procedure (power on)

If the reset vector is not valid, either the Apple IIc has just been turned on or something has caused memory contents to be changed. The reset routine clears the display and puts the string `Apple© IIc` at the top of the display. It loads the reset vector and the validity-check byte, then initiates the startup routine that resides in the disk controller firmware. The bootstrap routine then loads whatever operating system resides on the disk in the built-in drive. When the operating system has been loaded, it displays other messages on the screen. If there is no disk in the disk drive, the drive motor keeps spinning for a brief time. Then the firmware shuts it off and displays the message `Check Disk Drive` at the bottom of the screen.

If you press Control-Reset again before the startup procedure is completed, the reset routine continues without using the disk, and passes control to the Applesoft BASIC interpreter.

---

## The warm-start procedure (Control-Reset)

Whenever you press Control-Reset when the Apple IIc has already completed a cold-start reset, the reset vector is still valid and it is not necessary to reinitialize the entire system. The reset routine simply uses the vector to transfer control to the program it points to, which at power-up is the Applesoft interpreter.

If the vector does point to the Applesoft interpreter, your Applesoft program and variables are still intact. If you are using DOS or ProDOS, that operating system is the resident program and it restarts the BASIC interpreter you were using when you pressed Control-Reset.

---

<b>Important</b>	A program residing only in bank-switched RAM cannot use the reset vector to regain control after a reset, because upon reset the hardware selects the ROM for reading in the bank-switched memory space.
------------------	--

---

---

## Forced cold start (Open Apple-Control-Reset)

If a program has set the reset vector to point to its own warm-start address, as described below, pressing Control-Reset causes transfer of control to that program. If you want to stop such a program without turning the power off and on, you can force a cold-start reset by holding down Control and Open Apple, then pressing and releasing Reset.

---

<b>Important</b>	When you want to stop a program unconditionally—for example, to start up the Apple IIc with some other program—you should use the forced cold-start reset, Open Apple-Control-Reset, instead of turning the power off and on.
------------------	---

---

---

<b>UniDisk 3.5</b>	You must hold Open Apple down until the built-in drive starts to spin. If you release Open Apple before the drive starts to spin, the Apple IIc drops into BASIC instead of rebooting.
--------------------	--

---

The forced cold-start reset works as follows. First, it destroys the program or data in memory by writing two bytes of arbitrary data into each page of main RAM. The two bytes that get written over in page \$03 are the ones that contain the reset vector. The warm-start reset routine finds the error, and so performs a normal cold-start reset.

Note that if you press both Open Apple and Solid Apple during power-up or Control-Reset, built-in exercise code is executed. This code is for production and has no end-user value.

---

## The reset vector

The cold-start reset routine stores the starting address of the built-in Applesoft interpreter, low-order byte first, in the reset vector address at locations \$03F2 and \$03F3. It then stores a validity-check byte, also called the power-up byte, at location \$03F4. The validity-check byte is computed by performing an exclusive-OR of the second byte of the vector with the constant 165 (hexadecimal \$A5). Each time you reset the Apple IIc, the reset routine uses this byte to determine whether the reset vector is still valid.

You can change the reset vector so that the reset routine will transfer control to your program instead of to the Applesoft interpreter. For this to work, you must also change the validity-check byte to the exclusive-OR of the high-order byte of your new reset vector with the constant 165 (\$A5). If you fail to do this, then the next time you reset the Apple IIc, the reset routine will determine that the reset vector is invalid and perform a cold-start reset, eventually transferring control to the disk bootstrap routine or to Applesoft.

There is a subroutine that generates the validity-check byte for the current reset vector. This subroutine, called *SetPWRC*, is at location \$FB6F. When your program finishes, it can return the Apple IIc to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.





## **Chapter 3**



# **Introduction to Apple IIc I/O**

This chapter is an introduction to the built-in I/O capabilities of the Apple IIc. It outlines

- standard I/O links and their functions
- I/O firmware protocols
- dedicated memory storage locations
- direct I/O

The next six chapters discuss these capabilities in detail.

---

---

## The standard I/O links

You can use some of the routines in the Apple IIc's firmware for your own programs. This can save you both program space and the time and effort of writing all your own I/O routines.

To use the built-in firmware routines, your program must perform a JSR to the routine's entry address. The called routine then performs an indirect jump through an address stored somewhere in RAM and begins executing. When the routine has finished doing its work, it returns (with an RTS) to your program at the first instruction following the JSR used to call the routine. Memory locations used for transferring control to other subroutines, such as the indirect jump's address used by the character I/O routine, are sometimes called *vectors*. In this manual, the locations used for transferring control to the Apple IIc's I/O subroutines are called the *I/O links*.

In an Apple IIc running without an operating system, each I/O link normally contains the address of the standard input or output subroutine. An operating system will typically place addresses of its own I/O routines in these link locations instead.

By calling the I/O subroutines that then jump to the routines pointed to by the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly with other software, such as the operating system or a device driver. The I/O links contain the addresses of KeyIn and COut1 if the enhanced video firmware is off (when the display shows a flashing checkerboard cursor), and of C3KeyIn and C3COut1 if that firmware is on (when the display shows an inverse solid cursor).

The standard I/O links are two pairs of locations in the Apple IIc RAM in the range \$36 through \$39 that are used for controlling character input and output.

◆ *Note:* Not all operating systems use the standard I/O links. For example, Apple Pascal does not use them.

The link at locations \$36 and \$37 is called *CSW* (character output switch). Individually, location \$36 is called *CSWL* (CSW low) and location \$37 is called *CSWH* (CSW high). This link holds the starting address of the subroutine the Apple IIc is currently using for single-character output. This address is normally \$FDF0, the address of routine COut1.

The Monitor is discussed in Chapter 10.

When you issue either a PR#n from BASIC or an n Control-P from the Monitor, the Apple IIc changes this link address to the first address in the ROM space allocated to port n. That address has the form \$Cn00. Subsequent calls for character output are thus transferred to the firmware starting at that address. When it has finished, the firmware executes an RTS (return from subroutine) instruction to return control to the calling program. Sometimes a PR#n will cause both input and output switches to be changed (as in the 80-column firmware).

A similar link at locations \$38 and \$39 is called *KSW* (keyboard input switch). Individually, location \$38 is called *KSWL* (KSW low) and location \$39 is called *KSWH* (KSW high). This link holds the starting address of the routine currently being used for single-character input—normally \$FD1B, the starting address of the standard input routine KeyIn.

When you issue an IN#n command from BASIC or an n Control-K from the Monitor, the Apple IIc changes the link address in KSW to \$Cn00, the beginning of an I/O firmware subroutine. Subsequent calls for character input are thus transferred to that firmware. The firmware puts the input character, with its high bit set, into the accumulator and executes an RTS (return from subroutine) instruction to return control to the program that requested input.

When a disk operating system (DOS or ProDOS) is running, one or both of the standard I/O links hold addresses of the disk operating system's input and output routines. The operating system has internal locations that hold the addresses of the currently active character input and output routines.

---

**Warning**

If a program that is running with DOS or ProDOS changes the standard link addresses, either directly or via IN# and PR# commands, the operating system may be disconnected from the system. To avoid this problem, when programming in BASIC you should always issue an empty PRINT statement (to be sure that what follows begins a new line) before issuing the PRINT statement containing Control-D and the IN# or PR# command.

---

Refer to the section on input and output link addresses in the operating system manuals for further details.

GetLn also provides on-screen editing features. See "Editing With GetLn."

After changing either CSW or KSW, your assembly-language programs running under DOS should call the subroutine at location \$03EA. This subroutine transfers the link address to a location inside the operating system and then restores the operating system link address in the standard link location.

---

---

## Standard input features

The Apple IIc's firmware includes two different subroutines for reading from the keyboard, *RdKey* (read key) and *GetLn* (get line).

RdKey calls the current character input routine (that is, the one whose address is stored at KSW). This is normally KeyIn or C3KeyIn, which accepts one character from the keyboard. GetLn accepts a *sequence* of characters terminated with a carriage return. Thus GetLn allows line-oriented input using the current input routine.

---

### RdKey subroutine

A program can get a character from the keyboard by making a subroutine call to RdKey at memory location \$FD0C. RdKey passes control via the input link KSW to the current input subroutine, which is normally KeyIn.

RdKey displays a cursor at the current cursor position, which is immediately to the right of whatever character you last sent to the display (normally by using the COut routine, described below).

---

### KeyIn subroutine

KeyIn is the standard input subroutine. When your program calls it, KeyIn displays a cursor, waits until someone presses a key, then inserts the ASCII code of the key just pressed in the accumulator and returns to the calling program.

If the enhanced video firmware is inactive, KeyIn displays a cursor by alternately storing a checkerboard block in the cursor location, storing the original character, then storing the checkerboard again. If the firmware is active, C3KeyIn places a block cursor on the screen by inverting (swapping black for white) the character at the cursor position.

KeyIn also generates a random number. While it is waiting for the user to press a key, KeyIn repeatedly increments the 16-bit number in memory locations \$4E and \$4F. This number keeps increasing from 0 to \$FFFF (65535), then starts over again at 0. The value of this number changes so rapidly that it is very difficult to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a pseudo-random number routine.

---

## GetLn subroutine

Programs often need strings of characters as input. While you could call RdKey repeatedly to get several characters from the keyboard, there is an easier way to do it. The routine that you want to use in this case is named *GetLn*, and it starts at location \$FD6A. Using repeated calls to RdKey, GetLn accepts characters from the standard input subroutine—usually KeyIn—and puts them into the input buffer located in the memory page from \$0200 to \$02FF. GetLn also provides you with some basic on-screen editing and control features.

The first thing GetLn does when you call it is to display a prompt. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. Table 3-1 shows the prompt characters used by different programs on the Apple IIc.

GetLn uses the character stored at memory location \$33 as the prompt character. In an assembly-language program, you can change the prompt to any character you wish. In BASIC, changing the prompt character has no effect because both BASIC interpreters and the Monitor restore it each time they request input from the user.

**Table 3-1**  
Prompt characters

Prompt character	Program requesting input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix D)
>	Integer BASIC (Appendix D)
*	Firmware Monitor (Chapter 10)

❖ *Note:* Applesoft uses GetLn1 (\$FD6F) when a program is executing. GetLn1 does not print a prompt.

As the user types each character, GetLn sends the character to the standard output routine—normally COut1—which displays it at the current cursor position and then advances the cursor to indicate the next character position. Control characters echoed by GetLn are not executed.

GetLn stores the characters in its buffer, starting at memory location \$0200 and using the X register to index the buffer. GetLn continues to accept and display characters until the user presses Return (or Control-X to cancel the line). Then it clears the remainder of the line the cursor is on, stores the carriage-return code to mark the end of the buffer, places the cursor at the beginning of the next line, and returns.

The maximum line-length that GetLn can handle is 255 characters. If the user types more than this, GetLn sends a backslash (\) and a carriage return to the display, cancels the line it has accepted so far, and starts over. To warn the user that the line is getting full, GetLn sounds a bell (tone) at every keypress after the 248th.

❖ *Note:* The Applesoft interpreter accepts only 239 characters.

---

## Escape codes with GetLn

GetLn has many special functions that you invoke by typing escape codes on the keyboard. An escape code is sent by pressing Escape, releasing it, and then pressing some other key, as shown in Table 3-2.

---

### Important

Be sure to release Escape right away. If you hold it too long, the auto-repeat mechanism begins, which may cancel the Escape.

---

**Table 3-2**  
Escape codes with GetLn

Escape code	Function
Escape	Clears the window and homes the cursor (places it in the upper-left corner of the screen); exits from escape mode
Escape A or Escape a	Moves the cursor right one line; exits from escape mode
Escape B or Escape b	Moves the cursor left one line; exits from escape mode
Escape C or Escape c	Moves the cursor down one line; exits from escape mode
Escape D or Escape d	Moves the cursor up one line; exits from escape mode
Escape E or Escape e	Clears to the end of the line; exits from escape mode
Escape F or Escape f	Clears to the bottom of the window; exits from escape mode
Escape I or Escape i or Escape Up Arrow	Moves the cursor up one line; remains in escape mode
Escape J or Escape j or Escape Left Arrow	Moves the cursor left one space; remains in escape mode*
Escape K or Escape k or Escape Right Arrow	Moves the cursor right one space; remains in escape mode*
Escape M or Escape m or Escape Down Arrow	Moves the cursor down one line; remains in escape mode*
Escape 4	Switches to 40-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 3-5); exits from escape mode†

**Table 3-2 (continued)**  
**Escape codes with GetLn**

Escape code	Function
Escape 8	Switches to 80-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 3-5); exits from escape mode†
Escape Control-D	Disables control characters; only carriage return, linefeed, bell, and backspace have an effect when printed
Escape Control-E	Reactivates control characters
Escape Control-Q	Deactivates the enhanced video firmware; sets links to KeyIn and COut1; restores normal window size (Table 3-5); exits from escape mode†

\* Cursor-control key; see text.

† This code functions only when the enhanced video firmware is active.

In escape mode, you can keep using the arrow keys and the cursor movement keys I, J, K, and M without pressing Escape again. This enables you to perform repeated cursor moves by holding down the appropriate key.

When GetLn is in escape mode, it displays an inverse plus sign as the cursor. You leave escape mode by typing any key other than a cursor movement key.

❖ *Note:* The escape codes with the arrow keys are the standard cursor movement keys on the Apple IIc. The escape codes with I, J, K, and M are the standard cursor movement keys on the Apple II and II Plus, and are present on the Apple IIc for compatibility.

Escape sequences can be used in the middle of an input line to change the appearance of the screen. They have no effect on the input line.

For an Introduction to editing with these features, refer to the *Applesoft Tutorial*.

---

## Editing with GetLn

Subroutine GetLn provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. Any program that uses GetLn for reading the keyboard has these features.

### Cancel line

Any time you are typing a line, pressing Control-X causes GetLn to cancel the line. GetLn displays a backslash (\) and issues a carriage return, then displays the prompt and waits for you to type a new line. GetLn takes the same action when you type more than 255 characters, as described above.

### Backspace

When you press Left Arrow (or Control-H), GetLn moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COut, which moves the cursor back one space. If you type another character now, it replaces the character you backspaced over, both on the display and in the line buffer.

Each time you press Left Arrow, it moves the cursor left and deletes another character, until you are back at the beginning of the line. If you then press Left Arrow one more time, you have effectively canceled the line, and GetLn issues a carriage return and displays the prompt. The cursor moves even if the deleted character is an invisible control character. Thus it is possible for screen alignment and buffer alignment to be different.

### Retype

Right Arrow (or Control-U) has a function that is complementary to the backspace function. When you press Right Arrow, GetLn picks up the character under the cursor just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display.

See "Escape Codes With GetLn."

---

---

## Standard output features

The standard output routine is named *COut* (character output). COut normally calls COut1 or C3COut1, which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COut1 and C3COut1 restrict their use of the display to an active area called the text window, described later in this chapter.

---

### COut subroutine

Your program makes a subroutine call to COut at memory location \$FDED with a character in the accumulator. COut then passes control via the output link CSW to the current output subroutine, normally COut1 or C3COut1, which takes the character in the accumulator and writes it out. If the accumulator contains an uppercase or lowercase letter, a number, or a special character, COut1 or C3COut1 displays it; if the accumulator contains a control character, COut1 or C3COut1 either performs one of the special functions described below or ignores the character.

Each time you send a character to COut1 or C3COut1, it displays the character at the current cursor position, replacing whatever was there, and then advances the cursor position one space to the right. If the cursor position is already at the right edge of the window, COut1 or C3COut1 moves it to the leftmost position on the next line down. If this would move the cursor position past the end of the last line in the window, COut1 or C3COut1 scrolls the display up one line and sets the cursor position at the left end of the new bottom line.

The cursor position is controlled by the values in memory locations \$24 and \$25. These locations are named CH, for cursor horizontal, and CV, for cursor vertical. COut1 and C3COut1 do not display a cursor, but the input routines described above do, and they use this cursor position. However, changing CV directly does not change the cursor's vertical position until the next carriage return or reaching the end of the current line causes a call to VTab (for setting the base address within windows). If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COut1 or C3COut1.

---

**Warning** When the video firmware is set for 80-column display, the value of CH is kept at 0 and the true horizontal position is stored at \$057B. When the 80-column video firmware is active, use \$057B instead of CH.

---

Escape codes are described under "Escape Codes With GetLn."

---

## Control characters with COut1

COut1 does not display control characters. Instead, the control characters listed in Table 3-3 are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked from the keyboard, either by typing the control character listed or by using the appropriate escape code. The stop-list function, described separately, can only be invoked from the keyboard.

**Table 3-3**  
Control characters with COut1

Control character	ASCII name	Apple IIc name	Action taken by COut1
Control-G	BEL	Bell	Produces a 1000-Hz tone for 0.1 second
Control-H	BS	Backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
Control-J	LF	Line feed	Moves cursor position down to next line in window; scrolls if needed
Control-M	CR	Return	Moves cursor position to left end of next line in window; scrolls if needed

---

## Control characters with C3COut1

When the 80-column firmware is active, COut calls C3COut1 instead of COut1 for character output. C3COut1 does not display control characters, but you can use some control characters to control some of what the routine does. All other control characters are ignored.

The control characters listed in Table 3-4 are used to initiate some action by the firmware. Except for the stop-list function (Control-S) you can send control characters to C3COut1 either from a program or from the Apple IIc's keyboard. The stop-list function can only be invoked from the keyboard. Most of the functions listed here can also be performed by using an equivalent escape code.

**Table 3-4**  
Control characters with C3COut1

Control character	ASCII name	Apple IIc name	Action taken by C3COut1
Control-G	BEL	Bell	Produces a 1000-Hz tone for 0.1 second
Control-H	BS	Backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
Control-J	LF	Line feed	Moves cursor position down to next line in window; scrolls if needed
Control-K	VT	Clear EOS	Clears from cursor position to the end of the screen*
Control-L	FF	Home and clear	Moves cursor position to upper-left corner of window and clears window*
Control-M	CR	Return	Moves cursor position to left end of next line in window; scrolls if needed
Control-N	SO	Normal	Sets display format normal*
Control-O	SI	Inverse	Sets display format inverse*
Control-Q	DC1	40-column	Sets display to 40-column*
Control-R	DC2	80-column	Sets display to 80-column*
Control-S	DC3	Stop-list	Stops listing characters on the display until another key is pressed†
Control-U	NAK	Quit	Turns off enhanced video firmware*
Control-V	SYN	Scroll	Scrolls the display down one line, leaving the cursor in the current position*
Control-W	ETB	Scroll-up	Scrolls the display up one line, leaving the cursor in the current position*

**Table 3-4 (continued)**  
Control characters with C3COut1

Control character	ASCII name	Apple IIc name	Action taken by C3COut1
Control-X	CAN	Disable MouseText	Disables MouseText character display; uses inverse uppercase
Control-Y	EM	Home	Moves cursor position to upper-left corner of window (but doesn't clear)*
Control-Z	SUB	Clear line	Clears the line the cursor position is on*
Control-[	ESC	Enable MouseText	Maps inverse uppercase characters to MouseText characters
Control-\	FS	Fwd. space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below*
Control-]	GS	Clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window)*
Control-__	US	Up	Moves cursor up a line, no scroll

\* Doesn't work from the keyboard.

† Only works from the keyboard.

## The stop-list feature

You can stop the Apple IIc from updating its display (if it is using either COut1 or C3COut1) by pressing Control-S. Whenever COut1 or C3COut1 gets a carriage return from the program, it checks the keyboard for a Control-S. If a Control-S has been pressed, COut1 or C3COut1 stops and waits for another key to be pressed before resuming. The character code of the key that is pressed is ignored unless it is Control-C, which is passed to the program. This feature lets you exit BASIC programs from stop-list mode.

---

## The text window

The active portion of the display is called the *text window*. After you start up the computer or perform a reset, the entire display is the text window. COut1 or C3COut1 puts characters only into the window; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can restrict video activity to any rectangular portion of the display by changing the current text window. Your programs can thus control the placement of text in the display and protect other portions of the screen from being written over by new text. To do this, store the appropriate values into four locations in memory to set the top, bottom, left margin, and width of the text window. The following memory locations control the text window:

- The left margin is stored in memory location \$20. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is 39 (hexadecimal \$27); in an 80-column display, the maximum value is 79 (hexadecimal \$4F).
- The width of the text window is stored in memory location \$21. For a 40-column display, this value is normally 40 (hexadecimal \$28); for an 80-column display, it is normally 80 (hexadecimal \$50).
- The position of the top line of the text window is stored in memory location \$22. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal \$17).
- The position of the bottom line of the screen plus 1 is stored in memory location \$23. It is normally 24 (hexadecimal \$18) for the bottom line of the display. Its minimum value is 1.

---

<b>Important</b>	Pascal does not use this method of supporting window widths.
------------------	--

---

---

<b>Warning</b>	Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80 columns). If this happens, COut1 or C3COut1 may put characters into memory locations outside the display page, possibly destroying programs or data.
----------------	--

---

Table 3-5 summarizes the memory locations and the possible values for the text window parameters.

**Table 3-5**  
Text window memory locations

Window parameter	Location		Minimum value		Normal values				Maximum values			
					40-col.		80-col.		40-col.		80-col.	
	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left edge	32	\$20	00	\$00	00	\$00	00	\$00	39	\$27	79	\$4F
Width	33	\$21	00	\$00	40	\$28	80	\$50	40	\$28	80	\$50
Top edge	34	\$22	00	\$00	00	\$00	00	\$00	23	\$17	23	\$17
Bottom edge	35	\$23	01	\$01	24	\$18	24	\$18	24	\$18	24	\$18

## Normal, inverse, and flashing text

The way that the Apple IIc displays characters is affected by two things: the value that is stored in the inverse flag (zero page location \$32), and whether the enhanced video firmware is off or on. The inverse flag's influence is discussed in the next two subsections.

These display character sets are described in Chapter 5.

If the enhanced video firmware is off, the Apple IIc displays what is called the *primary character set*; if the video firmware is on, the Apple IIc displays what is called the *alternate character set*.

The primary character set includes normal (light on dark), inverse (dark on light), and flashing (alternating normal and inverse) characters. Lowercase inverse characters are not included in the primary character set.

The alternate character set includes normal and inverse characters (including lowercase inverse), and a set of graphic characters called *MouseText*. Flashing characters are not included in the alternate character set.

If you want your program to display a character, it should first load the character to be displayed in the accumulator, and then call the character-output subroutine COut. For example, to display the character corresponding to \$C8, you can use something like this:

```
LDA #$C8
JSR COut
```

## Primary character set display

The primary character set is displayed by COut1, which operates only when the enhanced video firmware is off. The primary character set includes text in normal, inverse, or flashing format, but not inverse or flashing lowercase text.

If the value of the character sent to COut1 is greater than or equal to \$A0, that value is logically ANDed with the value of the inverse flag (at location \$32), then displayed. (If you're curious about which ASCII character is being sent, subtract \$80 from the value being sent to COut1.) You can use the following inverse flag values:

- ☐ \$FF (decimal 255) produces the normal character format.
- ☐ \$3F (decimal 63) produces the inverse character format.
- ☐ \$7F (decimal 127) produces the flashing character format.

### Important

---

To avoid unusual character display results, use only the three values \$3F, \$7F, and \$FF.

---

COut1 interprets character values from \$80 through \$9F as control characters and tries to execute them.

Character values from \$00 through \$7F are all interpreted as display characters, not control characters.

## Alternate character set display

The alternate character set includes normal and inverse format characters and the MouseText graphic characters. You should use C3COut1, the standard output link when the enhanced video firmware is *active*, to display the alternate character set. Here are the rules for using the alternate character set:

- ☐ Control characters are not displayed. Characters sent to C3COut1 are interpreted as control characters if they are in the range \$00 through \$1F or \$80 through \$9F.
- ☐ Characters in the range \$20 through \$7F and \$A0 through \$FF are displayed.
- ☐ If inverse flag (location \$32) bit 7 is 1, the character is normal.
- ☐ If inverse flag bit 7 is 0, the character is inverse.
- ☐ If MouseText is off, characters \$40 through \$5F are remapped to the range \$00 through \$1F and are displayed as uppercase inverse characters.
- ☐ If MouseText is on, character values \$40 through \$5F are left unchanged, and the characters are displayed as MouseText.

For a brief explanation of logical functions, refer to Appendix H.

MouseText is described more fully in Chapter 5.

See "MouseText" in Chapter 5.

---

---

## Port I/O

The Apple IIc is a member of the Apple II family of computers; however, unlike the Apple II, II Plus, and IIe, the Apple IIc does not have peripheral connector slots. In place of these, it has **ports**—the equivalent of firmware interface cards installed in slots.

---

### Standard link entry points

To maintain compatibility with existing software and its protocols, each port's I/O firmware has the same standard entry points (\$Cn00) as its equivalent slot in another Apple II would have. Table 3-6 shows these equivalents, as well as listing the chapter where each port is described.

The section on the standard I/O links describes how and when these entry addresses are placed in CSW and KSW. For example, issuing PR#n or IN#n changes the output and input links, respectively, so that subsequent output or input is handled by the firmware starting at address \$Cn00, and thus goes to or comes from the selected device.

---

#### Memory expansion

The memory expansion version of the Apple IIc places the mouse at \$C700 and the memory expansion card at \$C400.

---

**Table 3-6**  
Port characteristics

Port	Entry point	Port connector	Use	Chapter
1	\$C100	Serial port 1	Printers	7
2	\$C200	Serial port 2	Communication	8
3	\$C300	Video connectors	Enhanced video firmware	5
4	\$C400	Mouse	Mouse	9
5	\$C500	Intelligent disk port devices		
6	\$C600	Disk drives	Built-in and external drives	6
7	\$C700	No device	Reserved	6

---

#### Important

The addresses shown in Table 3-6 are not entry points in the sense that you can send characters to be printed by sending them to JSR \$Cn00.

---

## Firmware protocol

The Apple IIc supports a standard firmware protocol that, in addition to the standard link address, provides a table of device identification and entry points to standard and optional firmware subroutines. The protocol is equivalent to the Pascal 1.1 firmware protocol in use on other Apple II's, and is outlined in Table 3-7.

**Table 3-7**  
Firmware protocol locations

Address	Value	Description
\$Cn05	\$38	Pascal firmware card/port identifier.
\$Cn07	\$18	Pascal firmware card/port identifier.
\$Cn0B	\$01	Generic signature byte of a firmware card/port.
\$Cn0C	\$ci	Device signature byte: i is an identifier (not necessarily unique).  c = device class (not all used on the Apple IIc): \$00 reserved \$01 printer \$02 hand control or other X-Y device \$03 serial or parallel I/O card/port \$04 modem \$05 sound or speech device \$06 clock \$07 mass-storage device \$08 80-column card/port \$09 network or bus interface \$0A special purpose (none of the above) \$0B-0F reserved
\$Cn0D	ii	\$Cnii is the initialization entry address (PInit).
\$Cn0E	rr	\$Cnrr is the read routine entry address (PRead) (returns character read in A register).
\$Cn0F	ww	\$Cnww is the write routine entry address (PWrite) (enters with character to write in A register).
\$Cn10	ss	\$Cnss is the status routine entry address (PStatus) (enters with request code in A register: 0 to ask "Are you ready to accept output?" or 1 to ask "Do you have input ready?").
\$Cn11	\$00	If additional address bytes follow; nonzero if not.

Each table begins with identification bytes (\$Cn05 through \$Cn0C). Then, starting with address \$Cn0D, each byte in the table represents the low-order byte of the entry-point address of a firmware routine. The high-order byte of each address is always \$Cn, where n is the port number. Your program uses these byte values to construct its own jump table for subroutine calls to the ports.

All port routines require, on entry, that the X register contain \$Cn and that the Y register contain \$n0.

All routines, on exit, return an error code in the X register (0 means no error occurred; 3 means the request was invalid). The carry bit in the program status register usually contains a reply to a request code (0 means no; 1 means yes).

All the Apple IIc ports except the disk port conform to this protocol. The disk port is described in Chapter 6.

---

## Port I/O space

By a convention used in other Apple II series machines, each port or slot has exclusive use of 16 memory locations set aside for data input and output. The addresses of these locations are of the form \$C080 + #n0, where n is the port or slot number. Table 3-8 lists the port I/O space used in the Apple IIc.

---

## Port ROM space

In the Apple II and IIe, one 256-byte page of memory space is allocated to each slot. This space is used for read-only memory (ROM or PROM on the interface card) with driver programs that control the operation of input/output devices, as outlined in Table 3-7. On the Apple IIc, this space is dedicated to port firmware. However, I/O ROM space in the Apple IIc is used as efficiently as possible, and there is not a strict correspondence between firmware for port n and the \$Cn00 space, except as regards entry points.

For more information, refer to the hardware page memory map in Appendix B.

**Table 3-8**  
Port I/O locations

Port	Locations
1	\$C090-\$C09F
2	\$C0A0-\$C0AF
6	\$C0E0-\$C0EF

---

## Expansion ROM space

The 2K-byte memory space from \$C800 to \$CFFF in the Apple IIc—called *expansion ROM space* on the Apple II, II Plus, and IIe—contains the enhanced video firmware and port and memory transfer subroutines. The Apple IIc, unlike the II, II Plus, or IIe, always has this space switched in.

---

## Port screen hole RAM space

There are 128 bytes of memory (64 in main memory, 64 in auxiliary memory) allocated to the ports, eight bytes per port, as shown in Table 3-9. These bytes are reserved for use by the system, except as described in Chapters 4 through 9.

**Table 3-9**  
Port screen hole memory locations

Base address	Ports						
	1	2	3	4	5	6	7
\$0478	\$0479	\$047A	\$047B	\$047C	\$047D	\$047E	\$047F
\$04F8	\$04F9	\$04FA	\$04FB	\$04FC	\$04FD	\$04FE	\$04FF
\$0578	\$0579	\$057A	\$057B	\$057C	\$057D	\$057E	\$057F
\$05F8	\$05F9	\$05FA	\$05FB	\$05FC	\$05FD	\$05FE	\$05FF
\$0678	\$0679	\$067A	\$067B	\$067C	\$067D	\$067E	\$067F
\$06F8	\$06F9	\$06FA	\$06FB	\$06FC	\$06FD	\$06FE	\$06FF
\$0778	\$0779	\$077A	\$077B	\$077C	\$077D	\$077E	\$077F
\$07F8	\$07F9	\$07FA	\$07FB	\$07FC	\$07FD	\$07FE	\$07FF

These addresses are unused bytes in the RAM reserved for text and low-resolution graphics displays, and hence they are sometimes called screen holes. These particular locations are not displayed on the screen and their contents are not changed by the built-in output routines. In other words, they are used by the output routines but they are not part of the video display.

---

**Warning** All the screen holes in auxiliary memory, and many of them in main memory, are reserved for special use by Apple IIc firmware—for example, to store initialization information. Do not use any locations marked **reserved** in this manual.

---

The way that port firmware uses these RAM locations and their addresses is covered in Chapters 4 through 10.

---

---

## Interrupts

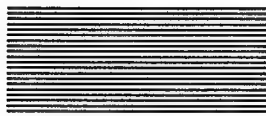
Appendix E describes interrupt handling on the Apple IIc.

Interrupts are a way to more efficiently use the hardware in a computer. Interrupt support built into the Apple IIc's firmware is described briefly below.

When the IRQ line on the 65C02 microprocessor is activated, the 65C02 transfers program control through the vector in locations \$FFFE through \$FFFF of ROM or whichever bank of RAM is switched in (Chapter 2). If ROM is switched in, this vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an interrupt that should be handled internally. If so, the Monitor handles it and then returns control to the interrupted program.

If the interrupt is due to a BRK (\$00) instruction, control is transferred through the BRK vector (\$03F0–\$03F1). Otherwise, control is transferred through the IRQ vector (\$03FE–\$03FF).





## **Chapter 4**

# **Keyboard and Speaker**

This chapter describes how to use two of the Apple IIc's built-in devices: the keyboard and the speaker.

---

---

## Keyboard input

Table 4-1 describes the characteristics of the keyboard that relate to programming. You won't have to write routines to read the keyboard from all your assembly-language programs since the Apple IIc firmware Monitor provides keyboard support through the three standard input routines described in Chapter 3—RdKey, KeyIn, and GetLn. You *can* do all your keyboard handling directly in your programs if you want to, but it's nice to know that you're not forced to.

For a description of how the keyboard strobe works, refer to Appendix E.

---

### Reading the keyboard

The keyboard encoder and ROM (see Chapter 11) can generate all 128 ASCII codes, so all the special character codes in the ASCII character set are available from the keyboard. Your machine-language programs can call RdKey to get characters from the keyboard. RdKey reads characters a byte at a time from the keyboard data location (\$C000) shown in Table 4-1.

Here is how your programs should go about reading the keyboard:

1. Test bit 7 of address \$C000 to see if a key has been pressed. Bit 7 is the keyboard strobe bit.
2. When bit 7 goes to a 1, you know that the low-order seven bits of \$C000 are a valid character.
3. Clear the keyboard strobe (bit 7) at \$C000 by reading or writing *anything* to address \$C010.

\$C010 has another function besides clearing the keyboard strobe: its high bit is a 1 while a key is pressed (except the Apple keys, Control, Shift, Caps Lock, and Reset). Bit 7 at this location is therefore called *any-key-down*. You could use this to let a program do something useful other than just waiting for the next key to be pressed. (People are generally a *lot* slower than the Apple IIc.) Check \$C010 occasionally to see if something should be done.

---

#### Important

If your program needs to read both the keyboard flag and the strobe, it must read the strobe bit first. Any time you read the any-key-down bit at \$C010, you also clear the keyboard strobe bit at \$C000.

---

**Table 4-1**  
Keyboard input characteristics

<b>Port number</b>	None	
<b>Commands</b>	Keyboard is always on, in the sense that any keypress generates a KSTRB.	
<b>Initial characteristics</b>	Reset routine clears the keyboard strobe and sets the keyboard as the standard input device (that is, sets KSW to point to RdKey).	
<b>Hardware locations</b>		
\$C000	Keyboard data and strobe	
\$C010	Any-key-down flag and clear-strobe switch	
\$C060	40-column switch status on bit 7; 1 = 40-column display = switch down	
\$C061	Open Apple status on bit 7; 1 = pressed (also game input switch 0)	
\$C062	Solid Apple status on bit 7; 1 = pressed	
<b>Monitor firmware routines</b>		
<b>Location</b>	<b>Name</b>	<b>Description</b>
\$FD6A	GetLn	Gets an input line with prompt
\$FD67	GetLnZ	Gets an input line with preceding carriage return
\$FD6F	GetLn1	Gets an input line, but with no preceding prompt
\$FD1B	KeyIn	The keyboard input subroutine
\$FD35	RdChar	Gets an input character or escape code
\$FD0C	RdKey	The standard character input subroutine
<b>Use of other pages</b>		
Page 2	The standard character string input buffer (see GetLn description)	

After your program has cleared the keyboard strobe, the strobe remains low until another key is pressed.

Table 4-2 shows the ASCII codes generated by all the keys on the Apple IIc keyboard. Remember, if the strobe bit is set, the character values that your program sees will be equal to the values given in Table 4-2 plus \$80.

**Table 4-2**  
Keys and ASCII codes

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
Delete	7F	DEL	7F	DEL	7F	DEL	7F	DEL
Left Arrow	08	BS	08	BS	08	BS	08	BS
Tab	09	HT	09	HT	09	HT	09	HT
Down Arrow	0A	LF	0A	LF	0A	LF	0A	LF
Up Arrow	0B	VT	0B	VT	0B	VT	0B	VT
Return	0D	CR	0D	CR	0D	CR	0D	CR
Right Arrow	15	NAK	15	NAK	15	NAK	15	NAK
Escape	1B	ESC	1B	ESC	1B	ESC	1B	ESC
Space	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C	,	2C	,	3C	<	3C	<
- _	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0 )	30	0	30	0	29	)	29	)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (	39	9	39	9	28	(	28	(
; :	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[ {	5B	[	1B	ESC	7B	{	1B	ESC
\	5C	\	1C	FS	7C		1C	FS
] }	5D	]	1D	GS	7D	}	1D	GS
! ~	60	!	60	!	7E	~	7E	~
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT

**Table 4-2 (continued)**  
Keys and ASCII codes

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Keystrokes can also generate interrupts. See Appendix E.

There are several keys that do not generate ASCII codes themselves, but alter the characters produced by other keys. These modifier keys are Control, Shift, and Caps Lock.

Your programs can also use the Open Apple and Solid Apple as character modifier keys while handling keyboard input, and, if one or both of them are pressed, branch to a special routine, such as a help program. Your program can read Open Apple at \$C061 and Solid Apple at \$C062.

Another key that doesn't generate a code is Reset, located at the upper-left corner of the keyboard; it is connected directly to the Apple IIc's processor. Pressing Reset with Control depressed normally causes the system to stop whatever program it's running and restart itself. If you hold Open Apple while pressing Control-Reset, the Apple IIc performs a forced cold start. The restart sequence is described in Chapter 2.

The reset routine is described in Chapter 2.

For information on how to have programs interpret keystrokes in a standard way, refer to the *Apple II Design Guidelines* listed in the Bibliography.

---

## Monitor firmware support for keyboard input

Chapter 3 describes the three standard Monitor input routines serving the keyboard: GetLn, RdKey, and KeyIn. This section discusses the three other available Monitor routines.

### GetLnZ

GetLnZ (at address \$FD67) is an alternate entry point for GetLn that first sends a carriage return to the standard output, then continues into GetLn.

### GetLn1

GetLn1 (at address \$FD6F) is an alternate entry point for GetLn that does not issue a prompt before it accepts the input line. However, if the user cancels the input line with too many backspaces or with Control-X, then GetLn1 issues the prompt stored at location \$33 when it gets another line.

### RdChar

RdChar (at address \$FD35) is a subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 3.

If the enhanced video firmware is active, Right Arrow (Control-U) reads a character from the screen as if it were typed from the keyboard. This is a function of the Monitor's built-in editing capability described in Chapter 3.

---

---

## Speaker output

Electrical specifications of the speaker circuit appear in Chapter 11.

The Apple IIc has a small speaker mounted near the front of the bottom plate of its case. The speaker is connected to a soft switch that toggles; that is, the switch has two states, off and on, and it changes from one to the other each time it is accessed. Table 4-3 describes the speaker output characteristics.

**Table 4-3**  
**Speaker output characteristics**

---

<b>Port number</b>	None.	
<b>Commands</b>	Some programs sound the speaker in response to Control-G.	
<b>Initial characteristics</b>	Reset routine sounds the speaker.	
<b>Hardware location</b>		
<b>\$C030</b>	Toggle speaker (read only).	
<b>Monitor firmware routines</b>		
<b>Location</b>	<b>Name</b>	<b>Description</b>
<b>\$FBDD</b>	Bell1	Sends a beep to the speaker.
<b>\$FF3A</b>	Bell	Sends Control-G to the current output.

---

## Using the speaker

If you switch the speaker once, by reading or writing to \$C030, it emits a click; to make longer sounds, access the speaker repeatedly. The switch for the speaker uses memory location \$C030. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program.

---

<b>Important</b>	You should always use a read operation to toggle the speaker. If you write to this soft switch, it switches twice in rapid succession. The resulting pulse is so short that the speaker doesn't have time to respond; it doesn't make a sound.
------------------	--

---

See Chapter 3.

---

## Monitor firmware support for speaker output

The Monitor supports the speaker with one simple routine, **Bell1**. A related routine, **Bell**, supports the current output device—the one that CSW points to.

### **Bell1**

**Bell1** (at address \$FDDB) makes a beep through the speaker by generating a 1-kHz tone in the Apple IIc's speaker for 0.1 second. This routine scrambles the A and X registers.

### **Bell**

The Monitor routine **Bell** (at location \$FF3A) writes a bell control character (ASCII Control-G) to the current output device. This routine leaves the accumulator holding \$87.



## **Chapter 5**



# **Video Display Output**

**NTSC** stands for *National Television Standards Committee*, a group that formulates broadcast and reception guidelines used by the USA and several other countries.

The Apple IIc's primary output device is its video display. You can use any ordinary color or monochrome video monitor with the Apple IIc. An ordinary monitor is one that accepts **NTSC**-compatible composite video. If you use Apple IIc color graphics with a black-and-white monitor, the display will appear as black, white, and two shades of gray.

If you are only using graphics modes and 40-column text, you can use a television set for your video display. If the TV set has an input connector for composite video, you can connect it directly to your Apple IIc; otherwise, you must attach an RF video modulator between the Apple IIc and the television set.

**Important**

The Apple IIc can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater.

Table 5-1 summarizes the video output port's characteristics and points to other information in this chapter.

**Table 5-1**  
Video output port characteristics

<b>Port number</b>	Output port 3.
<b>Commands</b>	See Figure 5-3.
<b>Initial characteristics</b>	See Figure 5-3. <i>Note:</i> If a program is to use the enhanced video firmware, it should turn it on and then immediately check the 80/40 switch. If the switch is in the 40 position, the program should issue a Control-Q.
<b>Hardware locations</b>	See Table 5-7.
<b>Monitor firmware routines</b>	See Table 5-11.
<b>I/O firmware entry points</b>	See Table 5-12.

---

---

## Video display specifications

Table 5-2 summarizes the video display's specifications, and provides a further guide to other information in this chapter.

**Table 5-2**  
Video display specifications

---

<b>Display modes</b>	40-column text; map: Figure 5-5
	80-column text; map: Figure 5-6
	Low-resolution color graphics; map: Figure 5-7
	High-resolution color graphics; map: Figure 5-8
	Double high-resolution color graphics; map: Figure 5-9
<b>Text capacity</b>	24 lines by 80 columns (character positions)
<b>Character set</b>	96 ASCII characters (uppercase and lowercase)
<b>Display formats</b>	Normal, inverse, flashing, MouseText (Table 5-3)
<b>Low-resolution graphics</b>	16 colors (Table 5-4): 40 horizontal by 48 vertical; map: Figure 5-7
<b>High-resolution graphics</b>	6 colors (Table 5-5): 140 horizontal by 192 vertical (restricted)
	Black and white: 280 horizontal by 192 vertical; map: Figure 5-8
<b>Double high-resolution graphics</b>	16 colors (Table 5-6): 140 horizontal by 192 vertical (no restrictions)
	Black and white: 560 horizontal by 192 vertical; map: Figure 5-9

The video signal produced by the Apple IIc is NTSC-compatible composite color video available at two places on the back panel of the Apple IIc: the RCA-type phono jack and the 15-pin D-type connector. Use the RCA-type phono jack to connect a video monitor, and the DB-15 connector for an external video modulator or other video expansion hardware.

See "Video Output Signals" in Chapter 11 for more on video expansion hardware.

---

---

## Text modes

Either of the Apple IIc's two text modes can display all 96 ASCII characters: uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide (with a few exceptions, such as underscore), leaving two blank columns of dots between characters in a row. Except for lowercase letters with descenders, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other monochrome color used by your monitor) dots on a dark background. Characters can also be displayed as black dots on a white background; this is called **inverse video**.

---

## Text character sets

The Apple IIc can display either of two text character sets: the *primary set* and an *alternate set* (Table 5-3). The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- ☐ normal, with white dots on a black screen
- ☐ inverse, with black dots on a white screen
- ☐ flashing, alternating between normal and inverse

The Apple IIc can display uppercase characters in all three formats—normal, inverse, and flashing—with the primary character set. Lowercase letters can only be displayed in normal format. This makes the primary character set compatible with most software written for the Apple II and II Plus, which can display text in flashing format but don't have lowercase characters.

The alternate character set trades the flashing format for a complete set of inverse characters. With the alternate character set, the Apple IIc can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format. It can also display MouseText.

See "MouseText."

To identify particular characters and values, refer to Table 4-2.

You can select between character sets with the alternate-text soft switch, described later in this chapter. Table 5-3 shows the character codes in decimal and hexadecimal for the Apple IIc primary and alternate character sets in normal, inverse, and flashing formats.

**Table 5-3**  
Display character sets

Hex values	Primary character set		Alternate character set	
	Character type	Format	Character type	Format
\$00-\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20-\$3F	Special characters	Inverse	Special characters	Inverse
\$40-\$5F	Uppercase letters	Flashing	MouseText	
\$60-\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80-\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0-\$BF	Special characters	Normal	Special character	Normal
\$C0-\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0-\$FF	Lowercase letters	Normal	Lowercase letters	Normal

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select format and the group within ASCII.

---

## MouseText

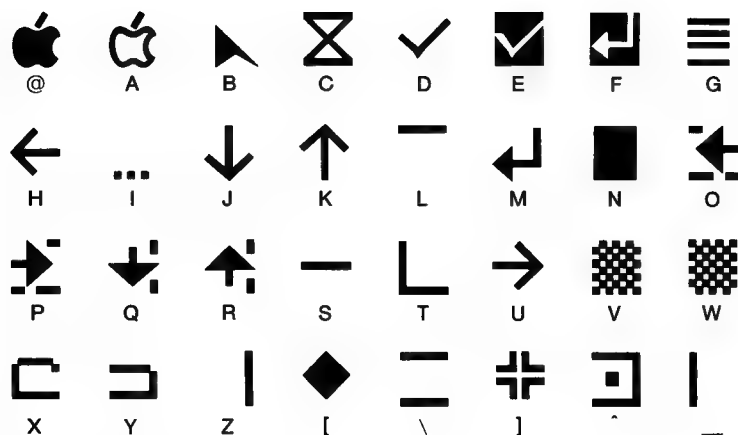
The alternate character set contains 32 graphics characters called *MouseText* in place of the primary set's inverse uppercase characters from \$40 through \$5F. These graphics are especially convenient to use with a mouse because they can be generated by character codes instead of groups of high-resolution byte values, and they can be moved around quickly. To use MouseText characters, do the following:

1. Turn on the enhanced video firmware with PR#3 or 6 Control-P.
2. Set inverse mode: use the INVERSE command or put \$3F in location \$32, or print Control-O.
3. Turn on MouseText with PRINT CHR\$(27); or pass \$1B to COut in the accumulator.
4. Print the uppercase letter (or other ASCII character in the range \$40 through \$5F:@[ \] ^ or \_ ) that corresponds to the MouseText character you want.
5. Turn off MouseText with PRINT CHR\$(24); or pass \$18 to COut1 in the accumulator.
6. Set normal mode: use the NORMAL command or put \$FF in location \$32, or print a Control-N.

Here is a sample Applesoft program that prints all the MouseText characters:

```
10 D$=CHR$(4)
20 PRINT PRINT D$;"PR#3"
30 INVERSE
40 PRINT CHR$(27);"ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_";
50 PRINT CHR$(24);
60 NORMAL
```

MouseText characters and their corresponding ASCII characters are shown in Figure 5-1.



**Figure 5-1**  
MouseText characters

---

## 40-column versus 80-column text

The Apple IIc has two text display modes: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare the two displays in Figure 5-2. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

```

]LIST 0,100

10  REM  APPLESOFT CHARACTER DEMO

20  TEXT : HOME
30  PRINT : PRINT "Applesoft Char
    acter Demo"
40  PRINT : PRINT "Which characte
    r set--"
50  PRINT : INPUT "Primary (P) or
    Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
65  LET A$ = LEFT$ (A$,1)
70  IF A$ = "P" THEN  POKE 49166,
    0
80  IF A$ = "A" THEN  POKE 49167,
    0
90  PRINT : PRINT "...printing th
    e same line, first"
100 PRINT " in NORMAL, then INVE
    RSE ,then FLASH:": PRINT

```

```

]LIST

```

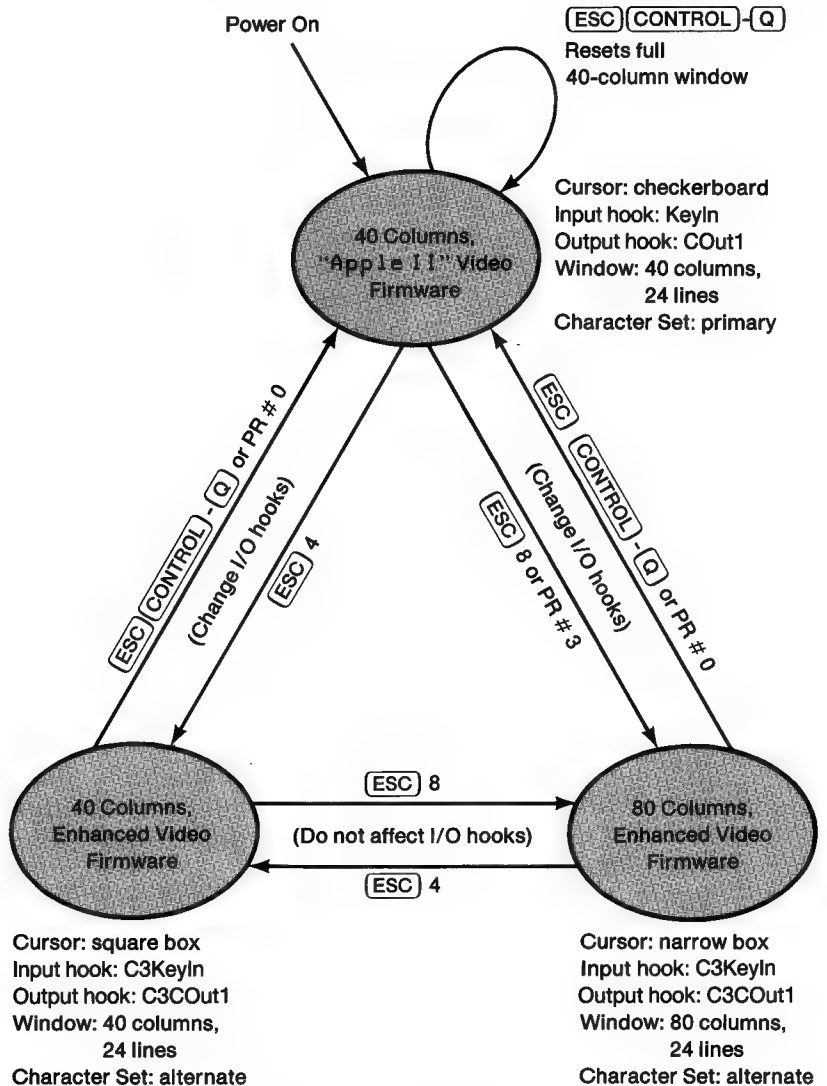
```

10  REM APPLESOFT CHARACTER DEMO
20  TEXT : HOME
30  PRINT : PRINT "Applesoft Character Demo"
40  PRINT : PRINT "Which character set--"
50  PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
70  LET A$ = LEFT$ (A$,1)
80  IF A$ = "P" THEN POKE 49166,0
90  IF A$ = "A" THEN POKE 49167,0
100 PRINT : PRINT "...printing the same line, first"
150 PRINT " in NORMAL, then INVERSE ,then FLASH:": PRINT
160 NORMAL : GOSUB 1000
170 INVERSE : GOSUB 1000
180 FLASH : GOSUB 1000
190 NORMAL : PRINT : PRINT : PRINT "Press any key to repeat." GET A$
200 GOTO 10
1000 PRINT : PRINT "SAMPLE TEXT: Now is the time--12:00"
1100 RETURN
]■

```

**Figure 5-2**  
40-column and 80-column text with alternate character set

Figure 5-3 shows the characteristics of the text display modes and how to switch between them.



**Figure 5-3**  
Text mode characteristics and switching

---

---

## Graphics modes

The Apple IIc can produce color video graphics in any of three different modes:

- low-resolution graphics, 48 rows by 40 columns
- high-resolution graphics, 192 rows by 280 columns
- double high-resolution graphics, 192 rows by 560 columns

Each graphics mode treats the screen as a rectangular array of spots. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes on the screen; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

**Table 5-4**  
Low-resolution graphics  
colors

Nibble value		
Dec	Hex	Color
0	\$00	Black
1	\$01	Magenta
2	\$02	Dark blue
3	\$03	Purple
4	\$04	Dark green
5	\$05	Gray 1
6	\$06	Medium blue
7	\$07	Light blue
8	\$08	Brown
9	\$09	Orange
10	\$0A	Gray 2
11	\$0B	Pink
12	\$0C	Light green
13	\$0D	Yellow
14	\$0E	Aquamarine
15	\$0F	White

*Note:* colors may vary, depending on adjustment of monitor or television set.

---

### Low-resolution graphics

The Apple IIc displays an array of 48 rows by 40 columns of colored blocks in the low-resolution graphics mode. Each block can be any one of sixteen colors, including black and white. On a black-and-white monitor or television set, these colors appear as black, white, and two shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

The low-resolution graphics display data are stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two low-resolution graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 5-4. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value \$D8 produces a brown block atop a yellow block on the screen.

As explained earlier in this chapter, the text display and the low-resolution graphics display use the same area in memory. Your programs should usually clear this part of memory when they change display modes, but you can store data as text and display them as graphics, or vice versa. All you have to do is change the mode switch, described later in this chapter, without changing the display data. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex low-resolution graphics displays quickly.

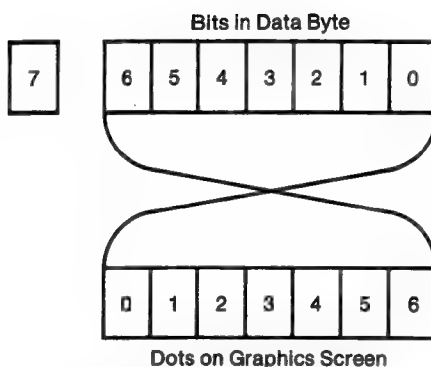
---

## High-resolution graphics

In the high-resolution graphics mode, the Apple IIc displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described below, by the color of adjacent dots. Adjacent dots of the same color merge to form a continuous colored area.

High-resolution graphics display data are stored in either of two 8192-byte areas in memory. These areas are called *high-resolution Page 1* and *Page 2*; think of them as display data buffers. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

The Apple IIc high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Apple IIc's memory. The seven low-order bits of each display byte control a row of seven adjacent dots on the screen, and 40 adjacent bytes in memory control a row of 280 (7 times 40) dots. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described below. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the next-least significant bit, and so on, as shown in Figure 5-4.



**Figure 5-4**  
High-resolution display bits

There is a simple correspondence between bits in memory and dots on the screen on a black-and-white monitor. A dot is white if the bit controlling it is on (1), and the dot is black if the bit is off (0). On a black-and-white television set, pairs of dots merge together; alternating black and white dots merge to a continuous gray.

A dot whose controlling bit is off (0) is black on an NTSC color monitor or a color television set. If the bit is on, the dot is white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the leftmost column of dots column 0, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control them are on, dots in even-numbered columns, 0, 2, 4, and so forth, are purple, and dots in odd-numbered columns are green—but only if the dots on either side are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange (again, only if the dots on either side are black). Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

In brief, high-resolution graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

- Dots in even-numbered columns can be black, purple, or blue.
- Dots in odd-numbered columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 5-5. The blacks and whites are numbered to remind you that the high-order bit is different.

**Table 5-5**  
High-resolution graphics colors

Bits 0–6	Bit 7 off	Bit 7 on
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

*Note:* Colors may vary, depending on adjustment of monitor or television set.

The peculiar behavior of the high-resolution colors reflects in part the way NTSC color television works. The dots that make up the Apple IIc video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating on and off dots at this spacing cause a color monitor or TV set to produce color, but two or more on dots together do not.

---

## Double high-resolution graphics

The horizontal resolution of double high-resolution graphics is 560 dots per line, with 192 lines. Double high-resolution graphics maps the low-order seven bits of the bytes in the two double high-resolution graphics pages. A double high-resolution page is made up of a 8192-byte page in main memory and an equivalent page having the same address in auxiliary memory. In most cases, only the first double high-resolution graphics page is used.

For more details about the way the Apple IIc produces color on a TV set, see Chapter 11. For a table of reversed bit patterns, refer to Appendix H.

The bytes in the main-memory and auxiliary-memory pages are displayed in exactly the same manner as the characters in 80-column text: of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed second. A dot whose controlling bit is off (0) is black when displayed.

Unlike high-resolution color, double high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a four-dot-wide window moving across the screen: at any given time, the color displayed corresponds to the 4-bit value from Table 5-6 that corresponds to the window's position (Figure 5-9). Effective horizontal resolution with color is 140 (560 divided by 4).

Table 5-6 describes the data values used to produce colors in double high-resolution graphics. To use the table, divide the column number by four and use the remainder to find the correct column: *ab0* is a byte residing in auxiliary memory corresponding to a remainder of 0 (byte 0, 4, 8, and so on), *mb1* is a byte residing in main memory corresponding to a remainder of 1 (byte 1, 2, 9 and so on), and similarly for *ab2* and *mb3*.

---

---

## Mixed-mode displays

Any of the graphics displays can have four lines of text, either 40-column or 80-column, at the bottom of the screen. Graphics displays with text at the bottom are called *mixed-mode displays*. To use them, the TEXT switch must be off (read \$C050) and the MIXED switch on (read \$C053).

---

<b>Important</b>	You cannot display 40-column text with double high-resolution graphics.
------------------	---

---

To determine what appears where in mixed-mode displays, refer to Figures 5-5 through 5-9 later in this chapter. See the bottom sixth of the appropriate text display (Figure 5-5 or 5-6) and the upper five-sixths (down to the heavy horizontal line) in the appropriate graphics display (Figures 5-7 to 5-9).

**Table 5-6**  
Double high-resolution graphics colors

Color	ab0	mb1	ab2	mb3	Repeated bit pattern
Black	\$00	\$00	\$00	\$00	0000
Magenta	\$08	\$11	\$22	\$44	0001
Brown	\$44	\$08	\$11	\$22	0010
Orange	\$4C	\$19	\$33	\$66	0011
Dark green	\$22	\$44	\$08	\$11	0100
Gray 1	\$2A	\$55	\$2A	\$55	0101
Green	\$66	\$4C	\$19	\$33	0110
Yellow	\$6E	\$5D	\$3B	\$77	0111
Dark blue	\$11	\$22	\$44	\$08	1000
Purple	\$19	\$33	\$66	\$4C	1001
Gray 2	\$55	\$2A	\$55	\$2A	1010
Pink	\$5D	\$3B	\$77	\$6E	1011
Medium blue	\$33	\$66	\$4C	\$19	1100
Light blue	\$3B	\$77	\$6E	\$5D	1101
Aqua	\$77	\$6E	\$5D	\$3B	1110
White	\$7F	\$7F	\$7F	\$7F	1111

*Note:* Colors may vary, depending on adjustment of monitor or television set.

---

---

## Display pages

The Apple IIc uses data stored in specific areas in memory to generate its video displays. These areas, called *display pages*, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object—a character, a colored block, or a group of adjacent dots—at a certain location on the display, depending on the current display mode.

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called *text Page 1* and *text Page 2*, and they are located at \$0400 through \$07FF and \$0800 through \$0BFF in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch between displays. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four lines of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages when the enhanced video firmware is active. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory. This additional memory is *not* the same as text Page 2—in fact, it is text Page 1X, and it occupies the same address space as text Page 1 (see Figure 2-11). The built-in firmware I/O routines described in Chapter 3 take care of this extra addressing automatically; that is one reason to use these routines for all normal text output.

---

**Important** The built-in video firmware always displays Page 1 text. You cannot write text to Page 2 with the built-in firmware.

---

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage as a low-resolution display, as shown in Table 5-7.

The double high-resolution graphics mode interleaves the two high-resolution pages (Pages 1 and 1X) in exactly the same way as 80-column text mode interleaves the text pages: column 0 and all subsequent even-numbered columns come from the auxiliary page; column 1 and all subsequent odd-numbered columns come from the main page.

**Table 5-7**  
Video display page locations

Display mode	Display page	Lowest address	Highest address
40-column text, low-resolution graphics	1	\$0400 1024	\$07FF 2047
	2*	\$0800 2048	\$0BFF 3071
80-column text	1	\$0400 1024	\$07FF 2047
	2*	\$0800 2048	\$0bFF 3071
High-resolution graphics	1	\$2000 8192	\$3FFF 16383
	2	\$4000 16384	\$5FFF 24575
Double high- resolution graphics	1†	\$2000 8192	\$3FFF 16383
	2†	\$4000 6384	\$5FFF 24575

\* This is not supported by firmware; for instructions on how to switch pages, refer to "Display Mode Switching."

† See "Double High-Resolution Graphics."

## Display mode switching

Table 5-8 shows the reserved locations for the soft switches that control the different display modes. The column of the table labeled *Action* indicates what to do to activate or read a switch setting: *R* means read the location, *W* means write anything to the location, *R/W* means read or write, and *R7* means read the location and then check bit 7.

Table 5-9 lists the display modes that the firmware can set up automatically. In the 40-column modes, the contents of the standard I/O hooks KSW and CSW (Chapter 3) determine whether the enhanced video firmware features are available or not. The firmware also takes care of setting or clearing AltChar.

Table 5-10 lists other display modes available but not supported by firmware. For modes that display Page 2 with the 80Col switch on, your program may have to turn 80Store off after the firmware has turned it on.

Double low-resolution shows on the display screen when HiRes is off and both 80Col and DHiRes are on. It is the low-resolution graphics equivalent of 80-column text, and it uses the same map (Figure 5-6), giving you 48 rows of 80 blocks.

The IOUDis (\$C07E) switch must be on to allow you to use locations \$C05E and \$C05F to change DHiRes. The firmware in fact leaves it on—and your program should, too—unless it wants to use locations \$C05E and \$C05F to change mouse values (Chapter 9).

**Table 5-8**  
Display soft switches

Name	Action	Hex	Function
AltChar	W	\$C00E	Off: Display text using primary character set
AltChar	W	\$C00F	On: Display text using alternate character set
RdAltChar	R7	\$C01E	Read AltChar switch (1 = on)
80Col	W	\$C00C	Off: Display 40 columns
80Col	W	\$C00D	On: Display 80 columns
Rd80Col	R7	\$C01F	Read 80Col switch (1 = on)
80Store	W	\$C000	Off: Cause Page2 on to select auxiliary RAM
80Store	W	\$C001	On: Allow Page2 to switch main RAM areas
Rd80Store	R7	\$C018	Read 80Store switch (1 = on)
Page2	R/W	\$C054	Off: Select Page 1
Page2	R/W	\$C055	On: Select Page 1X (80Store on) or 2
RdPage2	R7	\$C01C	Read Page2 switch (1 = on)
TEXT	R/W	\$C050	Off: Display graphics or (if MIXED on) mixed
TEXT	R/W	\$C051	On: Display text
RdTEXT	R7	\$C01A	Read TEXT switch (1 = on)
MIXED	R/W	\$C053	Off: Display only text or only graphics

**Table 5-8** (continued)  
Display soft switches

Name	Action	Hex	Function
MIXED	R/W	\$C054	On: (If TEXT off) display text and graphics
RdMIXED	R7	\$C01B	Read MIXED switch (1 = on)
HiRes	R/W	\$C057	Off: (If TEXT off) display low-resolution graphics
HiRes	R/W	\$C058	On: (If TEXT off) display high-resolution or (if DHiRes on) double high-resolution graphics
RdHiRes	R7	\$C01D	Read HiRes switch (1 = on)
IOUDis	W	\$C07E	On: Disable IOU access for addresses \$C058 to \$C05F; enable access to DHiRes switch
IOUDis	W	\$C07F	Off: Enable IOU access for addresses \$C058 to \$C05F; disable access to DHiRes switch*
RdIOUDis	R7	\$C07E	Read IOUDis switch (1 = off)†
DHiRes	R/W	\$C05E	On: (If IOUDis on) turn on double high-resolution
DHiRes	R/W	\$C05F	Off: (If IOUDis on) turn off double high-resolution
RdDHiRes	R7	\$C07F	Read DHiRes switch (1 = on)†

\* The firmware normally leaves IOUDis on. See also the following footnote.

† Reading or writing any address in the range \$C070–\$C07F also triggers the paddle timer and resets VBLint (Chapter 9).

**Table 5-9**

Display modes supported by firmware, including Applesoft

Display			Switches						
			80Col	80Store	Page2	TEXT	MIXED	HIRes	DHIRes
40-column	Text	1	Off		Off	On	Off	Off	Off
80-column	Text	1	On	*		On			
Low-res	Graphics	1	Off		Off	Off	Off	Off	Off
40/low	Mixed	1	Off		Off	Off	On	Off	
80/low	Mixed	1	On	*	Off	Off	On	Off	Off
Hi-res	Graphics	1	Off		Off	Off	Off	On	
Hi-res	Graphics	2	Off		On	Off	Off	On	
40/high	Mixed	1	Off		Off	Off	On	On	
80/high	Mixed	1	On	*	Off	Off	On	On	Off

\* 80Store is set by the firmware when 80Col is turned on.

**Table 5-10**

Other display modes

Display			Switches						
			80Col	80Store	Page2	TEXT	MIXED	HIRes	DHIRes
40-column	Text	2	Off		On	On			
80-column		2	On	Off	On	On			
Low-res	Graphics	2	Off		On	Off	Off	Off	
40/low	Mixed	2	Off		On	Off	On	Off	
80/low	Mixed	2	On	Off	On	Off	On	Off	Off
Dbl-low	Graphics	1	On	*	Off	Off	Off	Off	On
Dbl-low	Graphics	2	On	Off	On	Off	Off	Off	On
80/dbl-low	Mixed	1	On	*	Off	Off	On	Off	On
80/dbl-low	Mixed	2	On	Off	On	Off	On	Off	On
40/high	Mixed	2	Off		On	Off	On	On	
80/high	Mixed	2	On	Off	On	Off	On	On	Off
Dbl-high	Graphics	1	On	*	Off	Off	Off	On	On
Dbl-high	Graphics	2	On	Off	On	Off	Off	On	On
80/dbl-high	Mixed	1	On	*	Off	Off	On	On	On
80/dbl-high	Mixed	2	On	Off	On	Off	On	On	On

\* 80Store is set by the firmware when 80Col is turned on, and must be turned off to use the second 80-column or double high-resolution page. This means that you cannot use firmware routines such as COut when displaying Page 2 modes not supported by firmware.

For example, to switch to mixed 80-column and double high-resolution display Page 1, you can use these instructions in your program:

STA	\$C00D	Turns on 80Col; firmware then turns on 80Store.
LDA	\$C054	Turns off Page2; you could also have done a STA.
STA	\$C050	Turns off TEXT; that is, turns on graphics mode.
STA	\$C053	Turns on MIXED; it works now that TEXT is off.
STA	\$C057	Turns on HiRes; it works now that TEXT is off.
STA	\$C07E	Makes sure IOUDis is on so you can access DHiRes.
LDA	\$C05E	Turns on DHiRes; it works now that IOUDis is on.

---

---

## Display page maps

You should never have to store directly into display memory. Most high-level languages let you write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you should use the display features of the built-in I/O firmware.

---

### Warning

Never call any firmware with 80Col on or with 80Store and Page2 both on. If you do, the firmware will not function properly. As a general rule, always leave Page2 off.

---

All the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes (\$80 hex). For example, the first 128-byte block contains the data for rows 0, 8, and 16. The next 128-byte block contains data for rows 1, 9, and 17, and so on.

The display memory maps are shown in Figures 5-5 through 5-9. For a full description of the way the Apple IIc hardware handles display memory, see Chapter 11.

High-resolution graphics data are stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters.

The first 1024 bytes of the high-resolution display page contain the first row of dots from *each* of the 24 groups of eight rows of dots. The second 1024 bytes of the high-resolution display page contain the second row of dots from *each* group of eight rows of dots, and so on for all eight rows of all the groups. This fills up the 8192 bytes of the high-resolution display page.

The display maps show addresses only for each Page 1. To obtain addresses for text or low-resolution graphics Page 2, add 1024 (\$0400); to obtain addresses for high-resolution Page 2, add 8192 (\$2000).

The 80-column display works a little differently. Half of the data are stored in the normal text Page 1 memory, and the other half are stored in the *auxiliary* memory text Page 1. The display circuitry fetches bytes from the same address in both memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The characters in the even-numbered columns of the display are stored (starting with column 0) in main memory, and the characters in the odd-numbered columns of the display are stored (starting with column 1) in main memory.

For more details about the way the displays are generated, see Chapter 11.

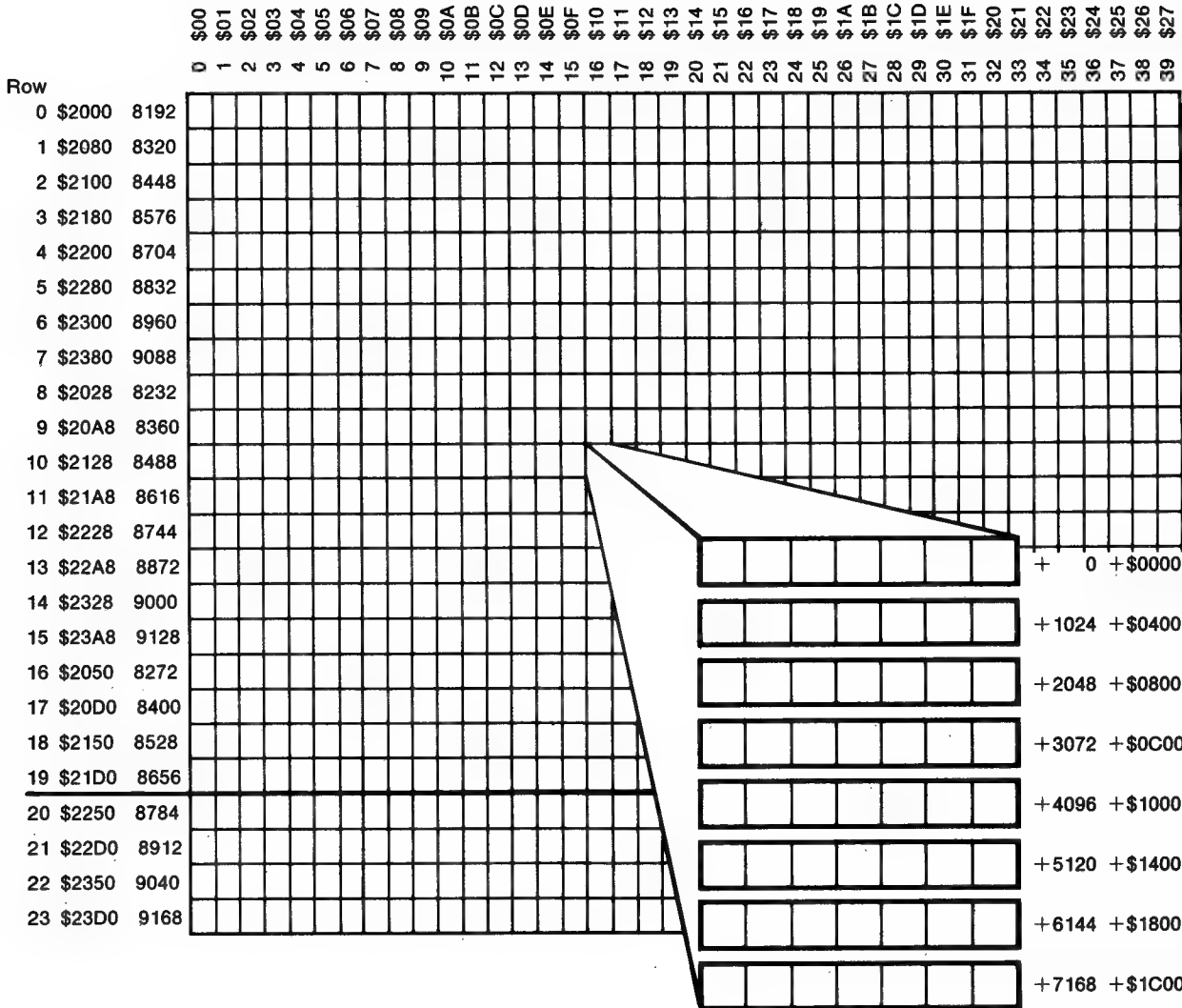
To store display data in auxiliary memory, first turn on the 80Store soft switch by writing to location \$C001. With 80Store on, the page-select switch Page2 selects between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in the auxiliary memory. To select auxiliary memory, turn the Page2 soft switch on by reading or writing at location \$C055.

The double high-resolution graphics display stores information in the same way as high-resolution graphics, except there is an auxiliary memory location as well as a main memory location corresponding to each address. The two sets of display information are interleaved in a manner similar to the interleaving of two 40-column displays to create an 80-column text display (Figure 5-9).

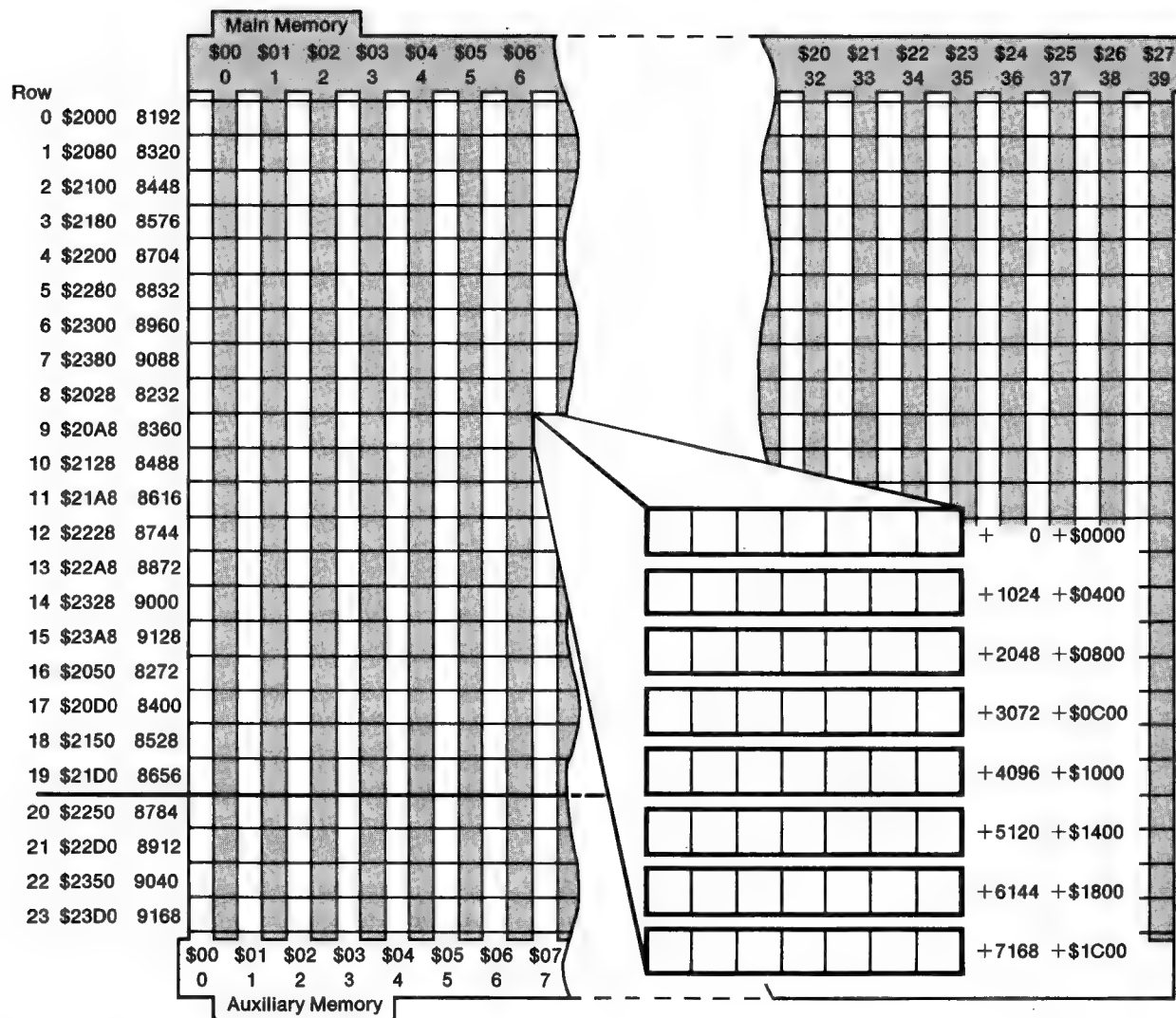








**Figure 5-8**  
Map of high-resolution graphics display



**Figure 5-9**  
Map of double high-resolution graphics display

## Monitor support for video display output

Table 5-11 summarizes the addresses and functions of the video display support routines the Monitor provides. Except for COut and COut1, which are explained in Chapter 3, these routines are described in the subsections that follow.

**Table 5-11**  
Monitor firmware routines

Name	Location	Description
ClrEOL	\$FC9C	Clears to end of line from current cursor position
ClEOLZ	\$FC9E	Clears to end of line using contents of Y register as cursor position
ClrEOP	\$FC42	Clears to bottom of window
ClrScr	F832	Clears the low-resolution screen
ClrTop	\$F836	Clears top 40 lines of low-resolution screen
COut	\$FDED	Calls output routine whose address is stored in CSW (normally COut1, Chapter 3)
COut1	\$FDF0	Displays a character on the screen (Chapter 3)
CROut	\$FD8E	Generates a carriage return character
CROut1	\$FD8B	Clears to end of line, then generates a carriage return character
HLine	\$F819	Draws a horizontal line of blocks
HOME	\$FC58	Clears the window and puts cursor in upper-left corner of window
PLOT	\$F800	Plots a single low-resolution block on the screen
PrBl2	\$F94A	Sends 1 to 256 blank spaces to the output device whose address is in CSW
PrByte	\$FDDA	Prints a hexadecimal byte
PrErr	\$FF2D	Sends ERR and Control-G to the output device whose output routine address is in CSW
PrHex	\$FDE3	Prints four bits as a hexadecimal number

**Table 5-11 (continued)**  
Monitor firmware routines

<b>Name</b>	<b>Location</b>	<b>Description</b>
PrntAX	\$F941	Prints contents of A and X in hexadecimal
SCRN	\$F871	Reads color value of a low resolution block on the screen
SetCol	\$F864	Sets the color for plotting in low resolution
VTabZ	\$FC24	Sets cursor vertical position (setting CV at location \$25 does not change vertical position until a carriage return)
VLine	\$F828	Draws a vertical line of low-resolution blocks

### **ClrEOL**

ClrEOL clears a text line from the cursor position to the right edge of the window. This routine destroys the contents of A and Y.

### **CIEOLZ**

CIEOLZ clears a text line to the right edge of the window, starting at the location given by base address BASL indexed by the contents of the Y register. This routine destroys the contents of A and Y.

### **ClrEOP**

ClrEOP clears the text window from the cursor position to the bottom of the window. This routine destroys the contents of A and Y.

### **ClrScr**

ClrScr clears the low-resolution graphics display to black. If you call this routine while the video display is in text mode, it fills the screen with inverse-mode at-sign (@) characters. This routine destroys the contents of A and Y.

### **ClrTop**

ClrTop is the same as ClrScr, except that it clears only the top 40 rows of the low-resolution display.

### **COut**

COut calls the current character output subroutine. The character to be sent to the output device should be in the accumulator. COut calls the subroutine whose address is stored in CSW (locations \$36 and \$37), usually the standard character output COut1.

### **COut1**

COut1 displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). It handles these control characters: carriage return, line feed, backspace, and bell. When it returns control to the calling program, all registers are intact.

### **CROut**

CROut sends a carriage return to the current output device.

### **CROut1**

CROut1 clears the screen from the current cursor position to the edge of the text window, then calls CROut.

### **HLine**

HLine draws a horizontal line of blocks of the color set by SetCol on the low-resolution graphics display. Call HLine with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLine returns with A and Y scrambled and X intact.

### **HOME**

HOME clears the display and puts the cursor in the upper-left corner of the screen.

### **PLOT**

PLOT puts a single block of the color value set by SetCol on the low-resolution display screen. Call PLOT with the vertical coordinate of the line in the accumulator, and its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.

### **PrBl2**

PrBl2 sends from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to send. If X = \$00, then PrBlank will send 256 blanks.

### **PrByte**

PrByte sends the contents of the accumulator in hexadecimal to the current output device. The contents of the accumulator are scrambled.

### **PrErr**

PrErr sends the word ERR, followed by a bell character (ASCII \$07), to the standard output device. On return, the accumulator is scrambled.

### **PrHex**

PrHex prints the lower nibble of the byte in the accumulator as a single hexadecimal digit. On return, the contents of the accumulator are scrambled.

### **PrntAX**

PrntAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte printed, and the X register contains the second. On return, the contents of the accumulator are scrambled.

### **SCRN**

SCRN returns the color value of a single block on the low-resolution display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. The block's color is returned in the accumulator. No other registers are changed.

### **SetCol**

SetCol sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 5-4.

### **VLine**

VLine draws a vertical line of blocks of the color set by SetCol on the low-resolution display. Call VLine with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLine returns with the accumulator scrambled.

---

---

## I/O firmware support for video display output

Apple IIc video firmware conforms to the I/O firmware protocol described in Chapter 3. However, it does not support windows other than the full 80-by-24 window in 80-column mode, and the full 40-by-24 window in 40-column mode.

The video (port 3) protocol table is shown in Table 5-12.

**Table 5-12**  
Port 3 firmware protocol table

Address	Value	Description
\$C30B	\$01	Generic signature byte of firmware cards
\$C30C	\$88	80-column card device signature
\$C30D	\$ii	\$C3ii is entry point of initialization routine (PInit)
\$C30E	\$rr	\$C3rr is entry point of read routine (PRead)
\$C30F	\$ww	\$C3ww is entry point of write routine (PWrite)
\$C310	\$ss	\$C3ss is entry point of the status routine (PStatus).

### PInit

PInit does the following:

- sets a full 80-column window
- sets 80Store (\$C001)
- sets 80Col (\$C00D)
- switches on AltChar (\$C00F)
- clears the screen; places cursor in upper-left corner
- displays the cursor

### PRead

PRead reads a character from the keyboard and places it in the accumulator with the high bit cleared. It also puts a 0 in the X register to indicate IOResult = GOOD.

## PWrite

PWrite should be called after placing a character in the accumulator with its high bit cleared. PWrite does the following:

- ☐ turns the cursor off
- ☐ if the character in the accumulator is not a control character, turns the high bit on for normal display or off for inverse display, displays it at the current cursor position, and advances the cursor; if at the end of a line, does carriage return but not line feed
- ☐ carries out control functions as shown in Table 5-13

**Table 5-13**  
Pascal video control functions

Control-	Hex	Function
E or e	\$05	Turns cursor on (enables cursor display)
F or f	\$06	Turns cursor off (disables cursor display)
G or g	\$07	Sounds bell (beeps)
H or h	\$08	Moves cursor left one column; if cursor was at beginning of line, moves it to end of previous line
J or j	\$0A	Moves cursor down one row; scrolls if needed
K or k	\$0B	Clears to end of screen
L or l	\$0C	Clears screen; moves cursor to upper-left position on screen
M or m	\$0D	Moves cursor to column 0
N or n	\$0E	Displays subsequent characters in normal video; characters already on display are unaffected
O or o	\$0F	Displays subsequent characters in inverse video; characters already on display are unaffected
V or v	\$16	Scrolls screen up one line; clears bottom line
W or w	\$17	Scrolls screen down one line; clears top line
Y or y	\$19	Moves cursor to upper-left (home) position on screen
Z or z	\$1A	Clears entire line that cursor is on

**Table 5-13 (continued)**  
Pascal video control functions

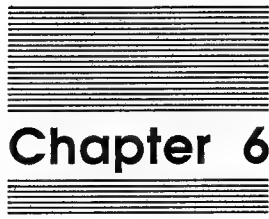
Control-	Hex	Function
or \	\$1C	Moves cursor right one column; if at end of line, does Control-M
} or	\$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor
^ or 6	\$1E	GOTOxy: Initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively
_	\$1F	If not at top of screen, moves cursor up one line

When PWrite has completed this, it

- ☐ turns the cursor back on (if it was not intentionally turned off)
- ☐ puts a 0 in the X register (IOResult = GOOD) and returns to the calling program

### **PStatus**

A program that calls PStatus must first put a request code in the accumulator: either a 0 (meaning "Ready for output?") or a 1 (meaning "Is there any input?"). PStatus returns with the reply in the carry bit: 0 (no) or 1 (yes). If the request was not 0 or 1, PStatus returns with a 3 in the X register (IOResult = ILLEGAL OPERATION); otherwise, PStatus returns with a 0 in the X register (IOResult = GOOD).



## **Chapter 6**

### **Block Device I/O**

A **block-type device**, or block device, executes I/O operations by grouping data into bundles, called *blocks*. A block may be made up of virtually any number of bytes, but in the Apple IIc a standard block is 512 bytes.

The Apple IIc supports both built-in and external **block-type devices**. External block devices may be 5.25-inch Disk IIc drives, UniDisk 3.5-inch disk drives, a memory expansion card, and other similar devices. If you use a 5.25-inch Disk IIc as an external drive, you must install it as the last device in the daisy chain.

Original IIc	The original Apple IIc does not support devices other than its internal 5.25-inch disk drive and an (optional) external 5.25-inch Disk IIc drive.
	The external block device interface is provided by the Smartport firmware. The Smartport is described later in this chapter.
UniDisk 3.5	The UniDisk 3.5 ROM contains an older version of the Smartport, the Protocol Converter. The description of the Smartport applies to the Protocol Converter, and vice versa.

The external disk drive connector is described under "Disk I/O" in Chapter 11.

## Disk drive I/O

Disk I/O firmware for the 5.25-inch drives resides in the \$C600 address space on the main side of the ROM. The built-in 5.25-inch drive is supported as if it were slot 6, drive 1, and the external 5.25-inch drive as if it were slot 6, drive 2.

Disk I/O firmware for the UniDisk 3.5 drive resides in the \$C500–\$C58D address space on the main side, and in the \$C880–\$CFFF address space on the auxiliary side of the ROM.

Table 6-1 summarizes the disk I/O port characteristics.

**Table 6-1**  
Disk I/O port characteristics

Port number	I/O port 6 drive 1 (built-in 5.25-inch drive). I/O port 6 drive 2 (external 5.25-inch drive). I/O port 5 drive 1 (external 3.5-inch drive).
Commands	IN#6 or PR#6 CALL –151 (to get to the Monitor from BASIC), then 6 Control-K or 6 Control-P.
Initial characteristics	All resets except Control-Reset with a valid reset vector eventually pass control to the built-in disk drive.

**Table 6-1 (continued)**  
Disk I/O port characteristics

---

<b>Hardware location</b> \$C0E0–EF	Reserved.
<b>Monitor firmware routines</b>	None.
<b>I/O firmware entry points</b>	\$C600 (port 6).
<b>Use of screen holes</b>	Port 6 main and auxiliary memory screen holes are reserved.

---

---

## Startup

The Apple IIc has two ways to start up—a cold start and a warm start. A cold start clears the machine's memory and tries to load an operating system from disk. A warm start halts the program that is running and leaves the machine in Applesoft with the contents of memory intact.

---

### Cold start

A cold start can be initiated by any of the following:

- ☐ turning the machine on
- ☐ pressing Open Apple-Control-Reset
- ☐ issuing a reboot command from the Monitor, BASIC, or a program
- ☐ pressing Control-Reset, if a valid reset vector does not exist

The startup routine first sets a number of soft switches to their initialization settings (see Chapter 2) and then passes control to the memory expansion card I/O entry point at \$C400. Because the contents of the memory expansion card's RAM are invalid in all cold-start situations, the Apple IIc cannot boot from card and control is returned to the startup routine.

---

<b>Original IIc</b>	The original Apple IIc does not support the memory expansion card; the restart routine in the original IIc begins with the internal 5.25-inch drive.
---------------------	--

---

When control is returned to the startup routine by the memory expansion card, it will attempt to boot the Apple IIc from the internal 5.25-inch drive. Control is passed to the 5.25-inch disk I/O entry point at \$C600. The code at this address turns on the internal drive motor, recalibrates the read/write head at track 0, then reads sector 0 from that track. The sector contents are loaded into main memory, starting at address \$0800. Once the contents of sector 0 have been loaded into main memory, control passes to \$0801. The program loaded depends on the operating system or application program on the disk in internal drive.

If for any reason the Apple IIc is unable to boot from the internal drive, control is returned to the startup routine. The startup routine then attempts to boot the Apple IIc from the external UniDisk 3.5 drive. Control is passed to the UniDisk 3.5 I/O entry point at \$C500, and the startup attempt proceeds in the same manner as that of the internal 5.25-inch drive.

---

<b>Original IIc</b>	<p>The original Apple IIc does not support the UniDisk 3.5 drive. However, it is possible to start the original Apple IIc from the external 5.25-inch drive. If you want to start your Apple IIc from the external 5.25-inch drive, you must use the ProDOS operating system. To start from the external drive, insert a ProDOS disk in the drive and</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> From the Monitor, type CALL -151 and press 7 Control-P.</li> <li><input type="checkbox"/> From BASIC, type PR#7.</li> </ul>
---------------------	---

---

To force a cold restart of the system:

- ☐ From BASIC, issue a PR#6 command.
  - ☐ From the Monitor, issue 6 Control-P.
  - ☐ From a machine-language program, JMP \$C600.
- 

<b>Memory expansion</b>	<p>To force a cold restart from a machine-language program in an Apple IIc that supports the memory expansion card, JMP \$C400 (the memory expansion card entry point).</p>
-------------------------	---

---



---

<b>UniDisk 3.5</b>	<p>The Apple IIc that supports the UniDisk 3.5 can force a cold restart that skips the internal 5.25-inch drive and passes control to the external drive port at \$C500 entry point. This allows the system to start up from the first <i>intelligent</i> drive connected to the external drive port. You can use the ProDOS or Pascal operating system if you want to start the system from an external drive, but DOS and versions of Pascal earlier than 1.3 will not work.</p>
--------------------	--

---

---

## Warm start

A warm start is initiated by pressing Control-Reset. The warm start routine checks \$F800–\$FFFF on the main side ROM for a valid reset vector. Provided a valid reset vector exists, control is turned over to the entry point specified by the vector. Generally, a warm start leaves you in BASIC with memory unchanged.

If there is no valid reset vector, a number of things may happen:

- ☐ The Apple IIc passes control to \$C600 on the main side ROM and the cold-start boot procedure begins.
- ☐ The Apple IIc beeps.
- ☐ The Apple IIc does nothing.

---

## Memory expansion

In the Apple IIc that supports the memory expansion card, control is turned over to \$C400 on the main side ROM in the event there is no valid reset vector.

---

---

---

## Memory expansion card I/O

The memory expansion card provides up to 1Mb of RAM, in 256K steps, for storage of program and data files. In this sense, it is like a very fast disk drive. Programs can be loaded into the memory expansion card's RAM, but in order to be executed they must be moved, in whole or in part, to the Apple IIc's main memory.

The memory expansion card is a block-type device, so I/O operations involving the card use the operating system or Smartport I/O interface. The Smartport I/O interface is described later in this chapter.

More information on the memory expansion card can be found in the *Apple IIc Memory Expansion Card Technical Reference*.

---

---

## The Smartport I/O interface

---

### Important

The rest of this chapter applies only to the UniDisk 3.5 and memory expansion versions of the Apple IIc.

---

---

**UniDisk 3.5** The Smartport and the Protocol Converter are essentially the same firmware interface with different names. All the specifications given in this manual for the Smartport interface apply to the Protocol Converter as well.

---

The rest of this chapter is about the Smartport, which is a set of assembly-language routines used to support external I/O devices, such as UniDisk 3.5. To ProDOS and Pascal 1.3, the Smartport appears to be a block device.

At the end of this chapter is an example of an assembly-language program that uses a Smartport call.

---

---

## Locating the Smartport

The Smartport code in the Apple IIc's firmware always begins at address \$C500. To ensure compatibility of your programs with the Apple IIc, however, your Smartport routines should always begin with a search for the Smartport. Your program can identify the Smartport by finding the following bytes:

```
$Cn01=$20
$Cn03=$00
$Cn05=$03
$Cn07=$00
```

where *n* can be an integer from 1 to 7. The Smartport entry point is then found at address  $\$Cn00 + (\$CnFF) + 3$ , where  $(\$CnFF)$  refers to the value of the byte located at  $\$CnFF$ . The sample program at the end of this chapter illustrates such a search.

---

**Important** The Smartport firmware is present even when the Memory Expansion Card is not. To check for the Memory Expansion Card, issue a STATUS call, code \$03, from the operating system or the Smartport. If the data returned indicates 0 bytes available, the card is not present.

---

On MLI calls, see the *ProDOS Technical Reference Manual*, Chapter 4.

## Issuing a call to the Smartport

Smartport calls are coded like ProDOS Machine Language Interface (MLI) calls: the program executes a JSR to a dispatch routine at address \$C500 + (\$C5FF) + 3, where (\$C5FF) refers to the value of the byte located at \$C5FF.

The Smartport call number and a two-byte pointer to the call's parameter list must immediately follow the call. Here is an example of a call to the Smartport:

```
IWMCALL
JSR DISPATCH      Calls PC command dispatcher
DFB CmdNum         Specifies the command type
DW  CmdList        2-byte (low, high) pointer to parameter list
BCS ERROR          Sets carry on an error
```

The command number (CmdNum) defines which Smartport call you want to make. Most Smartport calls include a two-byte pointer to a parameter list. The parameter list can contain information to be used by the call, or can provide space for information to be returned by the call. The length and content of the parameter list depend on the call being made. The format of each Smartport call's parameter list is described later in this chapter.

When the call has finished, the program resumes execution at the statement following the pointer to the parameter list. In the example above, the DFB and DW statements are skipped and execution resumes with the BCS statement. If the call is successful, the C flag (in the processor status register) is cleared (0), and the accumulator (the A register) is cleared to all 0's. If the call is unsuccessful, the C flag is set (1) and the error code is placed in the A register. After the Smartport call, the contents of the 65C02's registers are as follows:

Register	Processor status								X	Y	A	PC	S
	N	V	I	B	D	I	Z	C					
Successful call	x	x	1	u	0	u	x	0	x	x	0	JSR+3	u
Unsuccessful call	x	x	1	u	0	u	x	1	x	x	Error	JSR+3	u

x = undefined, except in cases where index information is returned in X and Y registers  
u = unchanged

On reading and writing to RAM, see "Bank-Switched Memory" in Chapter 4.

---

## Cautions

You must observe the following cautions when using the Smartport, *or your program will crash*:

- Leave space on the stack for the Smartport. The Smartport requires up to 35 bytes of stack space. Be sure to take this into account when calculating the stack space used by your program. If you don't do this, your program will fail if it tries to access data that *used* to be on the stack.
- Be sure that all RAM that you intend the Smartport to access is both read-enabled and write-enabled. The Smartport must be able to read from the RAM after writing to it, to obtain a checksum. Failure to observe this rule results in an error (BusErr \$06).
- Don't pass data to or from the Smartport through any zero page locations. Some of these locations are reserved for temporary storage of data by the Smartport, and your data will get changed.

---

---

## Descriptions of the Smartport calls

Calls to the Smartport are used

- to obtain status information about a device
- to reset a device
- to format the medium in a device
- to read from a device
- to write to a device
- to send control information to a device

The Smartport calls, in command-number sequence, are

STATUS (\$00)

Returns status information about a particular device, including general status (character or block device, read or write protection, format allowed, device on line); the device control block (set with the CONTROL call); the device newline status (character devices only); and device-specific information (number of blocks, ID string, device name, device type, device firmware version).

READ BLOCK (\$01)

Reads one 512-byte block from a disk device, and writes it to memory.

WRITE BLOCK (\$02)	Writes one 512-byte block from memory to a disk device.
FORMAT (\$03)	Prepares all blocks on a block device for reading and writing.
CONTROL (\$04)	Controls some device functions, including soft resets, setting the device control block (which controls global aspects of the device's operating environment), setting newline status (character devices only), and device interrupts. Several CONTROL calls are device-specific.
INIT (\$05)	Resets all resident devices. A global reset is done automatically on startup or system resets from the keyboard; an application should never have to reset all devices.
OPEN (\$06)	Prepares a character device for reading or writing.
CLOSE (\$07)	Tells a character device that a sequence of reads or writes is over.
READ (\$08)	Reads a specified number of bytes from a specified device.
WRITE (\$09)	Writes a specified number of bytes from memory to a specified device.

The following sections describe each Smartport call, including the command number, the parameter list, and error codes. The calls are discussed in command-number order in this format:

**Command name:** The name used to identify the call.

**Command number:** A hexadecimal number that specifies which call is being made to the Smartport.

**Parameter list:** A list of required call parameters.

**General description:** What the call does and what you use it for.

**Parameter descriptions:** A description of each parameter and the data it refers to. When a parameter refers to a status or control code, the meaning of each code number is discussed.

**Possible errors:** A list of the error codes that can be returned by this call. A complete list of Smartport error codes is included at the end of this chapter.

---

## STATUS

<b>Command number</b>	\$00
<b>Parameter list</b>	\$03 (parameter count) Unit number Status list pointer (low byte, high byte) Status code

The STATUS call returns status information about a specified device. The type of information returned is determined by the device and its status-code parameter. The status list pointer defines where the status information is returned to.

STATUS returns the number of bytes of status information that it generates in the X and Y registers, the low byte of this number in the X register, and the high byte in the Y register.

### Parameter descriptions

**Parameter count**

1-byte value    Three for this call.

**Unit number**

1-byte value    The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01–\$7E and are assigned according to the devices' positions in the chain.

---

<b>Important</b>	You can get the status of the Smartport itself if you use a unit number of \$00 and a status code of \$00 in a STATUS call (see the discussion beginning "Status code = \$00," below).
------------------	--

---

**Status list****pointer**

2-byte value      Points to the buffer to which the status is to be returned. The length required for the buffer varies depending on the status request being made.

**Status code**

1-byte value      Indicates what kind of status request is being made. Status codes are in the range \$00–\$FF, as follows:

Code	Status returned
\$00	Return device status
\$01	Return device control block (DCB) (not supported by UniDisk 3.5)
\$02	Return newline status (character devices only) (not supported by UniDisk 3.5)
\$03	Return device information block (DIB)
\$05	Return UniDisk 3.5 status

Status code = \$00 returns a device status consisting of four bytes. The first is the general status byte, with the following format:

Bit	Description
7	0 = character device, 1 = block device
6	1 = write allowed
5	1 = read allowed
4	1 = device on line or disk in drive
3	0 = format allowed
2	0 = medium write protected (block devices only)
1	1 = device currently interrupting
0	1 = device currently open (character devices only)

If the STATUS call is for a block device, the next three bytes (low byte first) are the size in 512-byte blocks. The maximum size is 16 million (\$FFFFFF) blocks (about 8 gigabytes). If the call is for a character device, these three bytes must be set to 0.

A STATUS call with status code = \$00 and unit number = \$00 returns the status of the Smartport itself. In this case, the status list consists of 8 bytes, as follows:

STAT_LIST	DFB	Number_Devices	Devices hooked to PC
	DFB	Interrupt_Status	Bit 6 clear = interrupt sent
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved
	DFB		Reserved

The Number\_Devices byte returns the total number of intelligent devices attached to the Smartport. The Interrupt\_Status byte is a copy of the asynchronous communications interface adapter (ACIA) status register at the time of the interrupt, and is used to indicate that a device requires interrupt servicing. If the sixth bit of this byte equals 0, one or more devices in the Smartport bus daisy chain must be serviced; your interrupt handler must poll each device on the chain to determine which ones.

- ❖ *About interrupts:* Devices that require interrupt servicing must use the EXTINT line on the Apple IIc's external disk port connector to be supported by the Smartport.

For example, UniDisk 3.5 does not support this line, and so cannot generate interrupts to the Smartport. See the description of the CONTROL command for instructions on enabling Smartport interrupts. See Appendix E for more information about programming with interrupts.

Status code = \$01 returns the device control block (DCB). The DCB is used to control various operating characteristics of a device and is device dependent. Each device has a default DCB, which can be altered with a CONTROL call. The first byte (the count byte) gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 256 bytes (257 including the count byte). Note that UniDisk 3.5 has no DCB and returns an error (BadCtl \$21) in response to this call.

Status code = \$02 returns newline status. Newline status applies only to character devices. A status code = \$02 passed to a block device returns a BadCtl (\$21) error.

On newline read mode, see  
Chapter 4 in the *ProDOS*  
*Technical Reference Manual*.

Status code = \$03 returns the device information block (DIB). The device's information block identifies the device, its type, and various other attributes. The returned status list has the following form:

STAT_LIST	DFB	Device_Statbyte1	Same as byte 1 in status code = 0
	DFB	Device_Size_Lo	Number of blocks (block device)
	DFB	Device_Size_Med	Number of blocks (middle byte)
	DFB	Device_Size_Hi	Number of blocks (high byte)
	DFB	ID_String_Length	Length in bytes (16 max.)
	ASC	'<device name>'	7-bit ASCII, uppercase, padded with spaces, 8th bit always=0 (16 bytes)
	DFB	Device_Type_Code	
	DFB	Device_Subtype_Code	
	DW	Version	Device firmware version number

Status code = \$05 returns the UniDisk 3.5 status. This call allows a diagnostic program to get more detailed information about the cause of a read or write error, and to examine the contents of the 65C02's registers after a CONTROL call with control code = \$05. The returned status list has this form:

STAT_LIST	DFB	\$00	
	DFB	Error	Soft Error byte (see below)
	DFB	Retries	Number of retries (see below)
	DFB	\$00	
	DFB	A_Value	Acc value after a CONTROL EXECUTE call
	DFB	X_Value	X value after EXECUTE
	DFB	Y_Value	Y value after EXECUTE
	DFB	P_Value	Processor status value after EXECUTE

The Error byte returned by a STATUS call with status code = \$05 contains the following bits:

Bit	Description
7	0
6	0
5	1 = address field mark or checksum error
4	1 = data field checksum error
3	1 = data field bit-slip mark mismatch
2	1 = seek error; unexpected track value found in address field
1	0
0	0

The Retries byte returned by a STATUS call with status code = \$05 specifies the number of address fields that had to be passed before the operation was completed. This information could be used, for example, to determine the number of passes necessary to read a data field correctly: If Retries is found to be greater than the number of sectors on the target track, then more than one pass was required.

The last four bytes of the status list are set only after a CONTROL call with control code = \$05, and are 0 after any other call (STATUS calls do not clear the status bytes).

### Possible errors

The following errors can be returned by the STATUS call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$21	BadCtl	Invalid status code
\$30-\$3F		Device-specific errors

---

## READ BLOCK

<b>Command number</b>	\$01
<b>Parameter list</b>	\$03 (parameter count) \$03 (parameter count) Unit number Data buffer (low byte, high byte) Block number (low byte, mid byte, high byte)

The READ BLOCK call reads one 512-byte block into memory from the block device specified by the unit-number parameter. The block of data is placed in a buffer starting at the address specified by the data-buffer parameter.

## Parameter descriptions

### Parameter count

1-byte value      Three for this call.

### Unit number

1-byte value      The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport.

### Data buffer

2-byte value      Points to the buffer into which the data are read. The buffer must be 512 or more bytes in length.

### Block number

3-byte value      The logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is 0 for all devices currently in use.)

## Possible errors

The following errors can be returned by the READ BLOCK call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

---

## WRITE BLOCK

**Command**      \$02  
**number**

**Parameter**    \$03 (parameter count)  
**list**            Unit number  
                  Data buffer (low byte, high byte)  
                  Block number (low byte, mid byte, high byte)

The WRITE BLOCK call writes one 512-byte block from memory to the disk device specified by the unit-number parameter. The block in memory starts at the address specified by the data-buffer parameter.

### Parameter descriptions

**Parameter**  
**count**

1-byte value      Three for this call.

**Unit number**

1-byte value      The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport.

**Data buffer**

2-byte value      Points to the buffer from which the data are to be written.

**Block number**

3-byte value      The logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is 0 for all devices currently in use.)

## Possible errors

The following errors can be returned by the WRITE BLOCK call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2B	NoWrite	Disk write protected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

---

## FORMAT

**Command number**      \$03

**Parameter list**      \$01 (parameter count)  
Unit number

The FORMAT call prepares all blocks on the recording medium of a block device for reading and writing. The formatting done by this call is specific to each device and is not linked to any operating system; for example, bitmaps and catalogs are not written by this call.

## Parameter descriptions

**Parameter count**

1-byte value      One for this call.

**Unit number**

1-byte value      The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport.

## Possible errors

The following errors can be returned by the FORMAT call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2B	NoWrite	Disk write protected
\$2F	OffLine	Device off-line or no disk in drive

---

## CONTROL

**Command number**     \$04

**Parameter list**     \$03 (parameter count)  
Unit number  
Control list (low byte, high byte)  
Control code

The CONTROL call sends control information to the device. The information can be of a general nature (such as resets or interrupts), or device-specific (such as Download to UniDisk 3.5 RAM).

---

**Important**     A CONTROL call to unit number \$00 sends control information to the Smartport itself. See the discussions of control code = \$00 and control code = \$01, below.

---

## Parameter descriptions

**Parameter count**

1-byte value     Three for this call.

**Unit number**

1-byte value     The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport. Use a unit number of \$00 in the CONTROL call to send control information to the Smartport itself.

**Control list**

2-byte value      Points to the buffer containing the control information. The first two bytes (the count bytes, low byte first) of the control list specify the number of bytes in the list (*not* including the count bytes); the remainder of the list contains the control information passed to the device.

---

**Important**      Every CONTROL call must have a control list; if no control information is being passed, then the control list consists of the count bytes only:

CTRL\_LIST DW \$00

---

**Control code**  
1-byte value

The number of the control request being made. Control codes are in the range \$00-\$FF. The following requests are not device specific:

**Code      Control function**

\$00	Reset the device
\$01	Set device control block (DCB)
\$02	Set newline status (character devices only)
\$03	Service device interrupt

Control requests to unit number \$00 are sent to the Smartport itself:

**Code      Control function**

\$00	Enable interrupts from Smartport
\$01	Disable interrupts from Smartport

Specific devices may respond to some or all of these additional control requests:

**Code      Control function**

\$04	Eject disk
\$05	Run a 65C02 subroutine
\$06	Set download address
\$07	Download to device RAM

Control code = \$00 performs a warm reset of the device and generally returns "housekeeping" values to some reset value. The control list for this call is device dependent.

The control list for this call for UniDisk 3.5 devices is

```
CTRL_LIST DW $00    No parameters are passed.
```

A CONTROL call with control code = \$00 and unit number = \$00 enables interrupts from the Smartport. This informs the firmware that external interrupts are possible, and directs it to call the user's interrupt handler if an interrupt occurs. It also turns on the ACIA for port 1.

When the user's interrupt handler identifies an external interrupt, you can determine if it came from the Smartport by making a STATUS call with unit number = \$00 and control code = \$00. See Appendix E for more information on handling interrupts.

Control code = \$01 alters the contents of the device control block (DCB). The DCB is used to set global aspects of a device's operating environment. Each device has a default setting for the DCB, set on initialization. Because the length of the DCB is device dependent, you should first read in the DCB with the STATUS call, then alter the bits of interest, and finally, use the same byte string as the control block for the CONTROL call. The first byte (the count byte) of the DCB gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 257 bytes, including the count byte.

Note that because UniDisk 3.5 has no DCB, a Set DCB CONTROL call to UniDisk 3.5 returns an error (BadCtl \$21).

A CONTROL call with control code = \$01 and unit number = \$00 disables interrupts from the Smartport. This call turns off the ACIA for port 1 and sets the least significant bit of the ACIA control register to 0.

Control code = \$02 sets a character device to newline enabled or newline disabled.

Control code = \$03 sends a device service interrupt. This code is to be used as needed for interrupt-driven devices.

Control code = \$04 ejects a disk. This code is to be used for devices that support an auto-eject feature. This code causes UniDisk 3.5 to auto-eject a disk. There are no parameters in the control list, and no errors are returned if the disk ejected correctly or there was no disk in the drive. Error code \$27 (IOError) is returned if the eject failed—that is, if a disk is still in the drive. The control list for UniDisk 3.5 is

CTRL\_LIST DW     \$00     No parameters are passed.

---

**Warning**     Control codes \$05 and higher are reserved; use of some of these codes can cause your system to crash.

---

### Possible errors

The following errors can be returned by the CONTROL call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$21	BadCtl	Invalid control code
\$22	BadCtlParm	Invalid parameter list
\$30–\$3F		Device-specific errors

---

### INIT

**Command number**     \$05

**Parameter list**     \$01 (parameter count)  
                             \$00 (unit number)

The INIT call resets all intelligent devices attached to the Smartport. The Smartport goes through an initialization sequence, cold-resetting all devices and sending each its unit number. This call is made automatically on startup; an application should never have to make this call.

## Parameter descriptions

### Parameter

#### count

1-byte value      One for this call.

#### Unit number

1-byte value      The unit number used in this call is always \$00.

## Possible errors

The following errors can be returned by the INIT call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$28	NoDrive	No device connected

---

## OPEN

**Command**      \$06  
**number**

**Parameter**      \$01 (parameter count)  
**list**              Unit number

The OPEN call prepares a character device for reading or writing.

Note that since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use an OPEN call with UniDisk 3.5 will result in an error (BadCmd \$01).

## Parameter descriptions

### Parameter

#### count

1-byte value      One for this call.

#### Unit number

1-byte value      The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport.

## Possible errors

The following errors can be returned by the OPEN call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$28	NoDrive	No device connected
\$2F	OffLine	Device off-line or no disk in drive

---

## CLOSE

**Command number**      \$07

**Parameter list**      \$01 (parameter count)  
Unit number

The CLOSE call tells a character device that a sequence of reads or writes is over.

Note that since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use a CLOSE call with UniDisk 3.5 will result in an error (BadCmd \$01).

## Parameter descriptions

**Parameter count**

1-byte value      One for this call.

**Unit number**

1-byte value      The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport.

## Possible errors

The following errors can be returned by the CLOSE call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$28	NoDrive	No device connected
\$2F	OffLine	Device off-line or no disk in drive

---

## READ

**Command number**     \$08

**Parameter list**     \$04 (parameter count)  
Unit number  
Buffer pointer (low byte, high byte)  
Byte count (low byte, high byte)  
Address pointer (low byte, mid byte, high byte)

The READ call reads into memory the number of bytes specified by the byte-count parameter. The bytes are placed in a buffer starting at the address specified by the buffer-pointer parameter.

## Parameter descriptions

**Parameter count**

1-byte value     Four for this call.

**Unit number**

1-byte value     The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01–\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Smartport.

**Buffer pointer**

2-byte point     Points to the buffer into which the data is read. The buffer must be large enough to contain the number of bytes requested by the byte-count parameter.

**Byte count**

2-byte value      Specifies the number of bytes to be transferred.

**Address****pointer**

3-byte value      Specifies the address to start reading from. The meaning of this parameter depends on the device being read.

**Possible errors**

The following errors can be returned by the READ call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

---

**WRITE**

**Command number**      \$09

**Parameter list**      \$04 (parameter count)-  
                          Unit number  
                          Buffer pointer (low byte, high byte)  
                          Byte count (low byte, high byte)  
                          Address pointer (low byte, mid byte, high byte)

The WRITE call writes from memory the number of bytes specified by the byte-count parameter to the specified unit. The bytes in memory start at the address indicated by the buffer-pointer parameter. The meaning of the address pointer depends on the type of device (see parameter descriptions).

## Parameter descriptions

### Parameter

#### count

1-byte value      Four for this call.

#### Unit number

1-byte value      The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range \$01-\$7E and are assigned according to the devices' positions in the daisy chain. A unit number of \$00 in the STATUS call returns the number of devices connected to the Protocol Converter.

#### Buffer pointer

2-byte value      Points to the buffer from which the data is to be written.

#### Byte count

2-byte value      Specifies the number of bytes to be transferred.

#### Address

#### pointer

3-byte value      Specifies the address to start writing from. The meaning of this parameter depends on the device being written to.

## Possible errors

The following errors can be returned by the WRITE call:

\$01	BadCmd	An unimplemented command was issued
\$04	BadPCnt	Bad call parameter count
\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off-line or no disk in drive

---

---

## An example: issuing a Smartport call

Here is an example of a program that issues a STATUS call to the Smartport to obtain information about a device.

The code for the Smartport in the version of the Apple IIc that supports UniDisk 3.5 always begins at address \$C500; however, to ensure compatibility with the Apple IIe, your programs should always do a search for the Smartport, as in this example.

```
0000:          1 *
0000:          2 *
0000:          3 *
0000:          4 * This example shows how to find
0000:          5 * and use a PC interface. A search
0000:          6 * is made for a PC, and when one is
0000:          7 * found, a vector is set up which
0000:          8 * points to the PC entry. Then a
0000:          9 * Device Information Block STATUS call
0000:         10 * is made, and if successful, the name
0000:         11 * string embedded in the DIB is output
0000:         12 * to the screen. Only the first device
0000:         13 * in the chain is accessed.
0000:         14 *
0000:         15 *
0000:         16             MSB    ON
0000:         17 *
0000:         18 *
0000:    0006    19 ZPTempl    equ    $0006    ;Temporary zero
0000:         20 *                      page storage
0000:    0007    21 ZPTempH    equ    $0007
0000:         22 *
0000:    FD8E    23 COut      equ    $FD8E    ;Console output
0000:    FD8E    24 CROut     equ    $FD8E    ;Carriage return
0000:         25 *
0000:    0000    26 StatusCmd  equ    0
0000:         27 *
0000:         28 *
0300:    0300    29             org    $300
0300:         30 *
0300:         31 * Find a Smartport in one of the
0300:         32 * slots.
0300:         33 *
0300:20 43 03    34             jsr    FindPC
0303:B0 1C    35             bcs    Error
0305:         36 *
0305:         37 * Now make the DIB call to the first guy
0305:         38 *
```

```

0305:20 67 03      39          jsr    Dispatch
0308:00            40          dfb    StatusCmd
0309:6A 03         41          dw     DParms
030B:B0 14 0321    42          bcs    Error
030D:             43      *
030D:             44      * Got the DIB; now print the name string
030D:             45      *
030D:A2 00        46          ldx    #0
030F:             47      morechars equ    *
030F:BD 74 03     48          lda    DIBName,x
0312:09 80        49          ora    #$80      ;COut wants high
0314:             50      *                      Bit set
0314:             51      * 0314:20 ED FD      52          jsr    COut
0317:E8           53          inx
0318:EC 73 03     54          cpx    DIBNameLen
031B:90 F2 030F   55          blt    morechars
031D:             56      *
031D:20 8E FD     57          jsr    CROut      ;Finish it off
0320:             58      *                      with a return
0320:             59      *
0320:60           60          rts
0321:             61      *
0321:             62      *
0321:             63      Error      equ    *
0321:             64      *
0321:             65      * There's either no PC around, or there
0321:             66      * was no Unit #1... give message
0321:             67      *
0321:A2 00        68          ldx    #0
0323:             69      err1      equ    *
0323:BD 2F 03     70          lda    Message,x
0326:F0 06 032E   71          beq    errout
0328:20 ED FD     72          jsr    COut
032B:E8           73          inx
032C:D0 F5 0323   74          bne    err1
032E:             75      *
032E:             76      errout    equ    *
032E:60           77          rts
032F:             78      *
032F:CE CF A0 D0  79      Message asc    'NO PC OR NO DEVICE'
0341:8D 00        80          dfb    $8D,0
0343:             81      *
0343:             82      *
0343:             83      FindPC    equ    *
0343:             84      *
0343:             85      * Search slot 7 to slot 1 looking for
0343:             86      * signature bytes
0343:             87      *
0343:A2 07        88          ldx    #7      ;Do for seven
0345:             89      *                      slots

```

```

0345:A9 C7      90      lda    #$C7
0347:85 07      91      sta    ZPTempH
0349:A9 00      92      lda    #$00
034B:85 06      93      sta    ZPTempL
034D:           94      *
034D:      034D  95  newslot  equ    *
034D:A0 07      96      ldy    #7
034F:           97      *
034F:      034F  98  again    equ    *
034F:B1 06      99      lda    (ZPTempL),y
0351:D9 70 03   100     cmp    sigtab,y      ;One of four
0354:           101     *                    byte signature
0354:F0 07      035D 102     beq    maybe      ;Found one
0356:           103     *                    signature byte
0356:C6 07      104     dec    ZPTempH
0358:CA         105     dex
0359:D0 F2      034D 106     bne    newslot
035B:           107     *
035B:           108     * If we get here, it's because we couldn't
035B:           109     * find a Smartport.
035B:           110     * Exit with the carry set.
035B:           111     *
035B:38         112     sec
035C:60         113     rts
035D:           114     *
035D:           115     * If we get here, it means that one or
035D:           116     * more of the signature bytes
035D:           117     * for this card are what we're looking
035D:           118     * for. Decrement the byte
035D:           119     * counter and branch back to verify any
035D:           120     * remaining bytes.
035D:           121     *
035D:      035D 122  maybe    equ    *
035D:88         123     dey
035E:88         124     dey      ;If N=1 then
035F:           125     *                    all sig bytes okay
035F:10 EE      034F 126     bpl    again
0361:           127     *
0361:           128     * Found a Smartport interface.
0361:           129     * Set up the call address.
0361:           130     * We already have the high byte ($CN);
0361:           131     * we just need the low byte.
0361:           132     *
0361:      0361 133  foundPC  equ    *
0361:A9 FF      134     lda    #$FF
0363:85 06      135     sta    ZPTempL
0365:A0 00      136     ldy    #0      ;For
0367:           137     *                    indirect load
0367:B1 06      138     lda    (ZPTempL),y    ;Get the
0369:           139     *                    byte

```

```

0369:          140 *
0369:          141 * Now the Acc has the low order ProDOS
0369:          142 * entry point. The PC entry is
0369:          143 * three locations past this...
0369:          144 *
0369:18         145         clc
036A:69 03    146         adc     #3
036C:85 06    147         sta     ZPTempL
036E:          148 *
036E:          149 * Now ZPTempL has the PC entry point.
036E:          150 * Return with carry clear.
036E:          151 *
036E:18         152         clc
036F:60       153         rts
0370:          154 *
0370:          155 *
0370:          156 * These are the PC signature bytes in
0370:          157 * their relative order.
0370:          158 * The $FF bytes are filler bytes and
0370:          159 * are not compared.
0370:          160 *
0370:FF 20 FF 00 161 sigtab    dfb     $FF,$20,$FF,$00
0374:FF 03 FF 00 162          dfb     $FF,$03,$FF,$00
0378:          163 *
0378:          164 *
0378:          0378 165 Dispatch equ     *
0378:6C 06 00    166          jmp     (ZPTempL) ;Simulate
037B:          167 * an indirect JSR to PC
037B:          168 *
037B:          169 *
037B:          037B 170 DParms    equ     *
037B:03         171 DPParmCt   dfb     3 ;Status
037C:          172 * calls have three parameters
037C:01         173 DPUnit     dfb     1
037D:80 03     174 DPBuffer   dw     DIB
037F:03         175 DPStatCode dfb     3
0380:          176 *
0380:          177 *
0380:          0380 178 DIB       equ     *
0380:00         179 DIBStatBytel dfb 0
0381:00 00 00   180 DIBDevSize dfb 0,0,0
0384:00         181 DIBNameLen dfb 0
0385:          0010 182 DIBName    ds     16,0
0395:00         183 DIBType     dfb     0
0396:00         184 DIBSubType dfb     0
0397:00 00     185 DIBVersion dw     0
0399:          186 *
0399:          187 *

```

## Summary of commands and parameters

The following is a summary of Smartport calls. In each case, byte 0 of the command parameter list (CmdLst) specifies the number of parameters in the command list (not including byte 0). Parameters that require more than one byte (the status list pointer, for example) are entered low byte first. The meaning of the address-pointer parameter is device specific. See the sections on the individual calls in this chapter for a discussion of each parameter.

Command	STATUS	READBLOCK	WRITEBLOCK	FORMAT	CONTROL
CmdNum	\$00	\$01	\$02	\$03	\$04
CmdList Byte					
0	\$03	\$03	\$03	\$01	\$03
1	Unit Num	Unit Num	Unit Num	Unit Num	Unit Num
2	Stat List Ptr	Buffer Ptr	Buffer Ptr		Ctl List Ptr
3					
4	Stat Code				Ctl Code
5		Block Num	Block Num		
6					

Command	INIT	OPEN	CLOSE	READ	WRITE
CmdNum	\$05	\$06	\$07	\$08	\$09
CmdList Byte					
0	\$01	\$01	\$01	\$04	\$04
1	\$00	Unit Num	Unit Num	Unit Num	Unit Num
2				Buffer Ptr	Buffer Ptr
3					
4				Byte Count	Byte Count
5					
6					
7				Address Ptr	Address Ptr
8					

Unused bytes

**Figure 6-1**  
Summary of Smartport calls

---

---


## Summary of error codes

The following is a summary of Smartport call error codes, including a brief description of the possible causes for each. If there is no error, the C flag (in the processor status register of the 65C02 microprocessor) is cleared (0) and the accumulator (the A register) contains 0s. If the call was unsuccessful, the C flag is set (1) and the A register contains the error code.


\$00		No error.
\$01	BadCmd	A nonexistent command was issued. Check the command number in the Smartport call.
\$04	BadPCnt	Bad call parameter count. The call parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.
\$06	BusErr	A communications error between the device controller and the host. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the host. Check for noise sources. Make sure the cable is properly shielded.
\$11	BadUnit	Unit number \$00 was used in a call other than STATUS, CONTROL, or INIT.
\$21	BadCtl	The control or status code is not supported by the device.
\$22	BadCtlParm	The control parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.

\$27	IOError	The device encountered an I/O error when trying to read or write to the recording medium. Make sure that the medium in the device is formatted and not defective and that the device is operating correctly.
\$28	NoDrive	The device is not connected. This can occur if the device is not connected but its controller is, or if there is no device with the unit number specified.
\$2B	NoWrite	The medium in the device is write protected.
\$2D	BadBlock	The block number is outside the range allowed for the medium in the device. Note that this range depends on the type of device and the type of medium in the device (single-sided versus double-sided disk, for example).
\$2F	OffLine	Device off-line or no disk in drive. Check the cables and connections. Make sure that the medium is present in the drive and that the drive is functioning correctly.
\$30–\$3F	DevSpec	Errors that differ from device to device. See the technical manual for the device in question for details. \$40–\$4F. Reserved for future expansion.
\$50–\$7F	NonFatal	A device-specific soft error. The operation completed successfully, but some exception condition was detected. See the technical manual for the device in question for details.





# Chapter 7



## Serial I/O Port 1

Serial port 1 is one of two serial I/O ports available on the Apple IIc. It is intended primarily as an output port for RS-232 devices, such as printers and plotters. It can be changed to a serial communication port (like port 2) by using the *System Utilities* disk or from a program.

---

**Warning** Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix F for a summary of these differences.

---

Table 7-1 summarizes the characteristics of this port if used as a printer/plotter port, and is a guide to the other information in this chapter. If you change port 1 to a communication port, refer to the descriptions in Chapter 8, and use 1 instead of 2 for the port number when required.

The serial port back panel connectors are described in Chapter 11.

**Table 7-1**  
Serial port 1 characteristics

---

<b>Port number</b>	Serial port 1.
<b>Commands</b>	Keyboard command: PR#1. BASIC command: PR#1. Monitor command: 1 Control-P (does not work if there is an operating system in RAM). All other commands: See Table 7-2.
<b>Initial characteristics</b>	See "Characteristics of Port 1 at Startup."
<b>Hardware page locations</b>	See Table 7-3.
<b>Monitor firmware routines</b>	None.
<b>I/O firmware entry points</b>	See Table 7-4.
<b>Use of screen holes</b>	See Table 7-5.
<b>Use of other pages</b>	None.

---

---

## Using serial port 1

You can access the firmware from BASIC in the usual way—that is, by issuing Control-D (if DOS or ProDOS is in RAM) and PR#1. Subsequent output is directed to the printer (or other device) connected to serial port 1.

To direct Pascal output to the printer, you can use either #6: or PRINTER:.

Your programs can also access the port by changing the value of CSW (see Chapter 3).

Table 7-2 lists the commands you can use with serial port 1, either from a program or from the keyboard, after you issue PR#1.

---

**UniDisk 3.5** Commands followed by an asterisk in Table 7-2 (with the exception of L) are available only on the version of the Apple IIc that supports UniDisk 3.5. These commands can be toggled by following them directly with E (enable) or D (disable).

---

Each command must be preceded by Control-I (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. You do not have to press Return after commands that you have entered from the keyboard, or send the return character from your program if it is sending commands to the port. You can type more than one command on a line, but each must be preceded by the command character.

**Table 7-2**  
Printer port commands

Command	Description																																				
nnn	Sets new line width of nnn (from 1 through 255).This command must be followed by N (see below) or by a carriage return.																																				
nnB	Sets baud rate to value corresponding to nn:																																				
	<table><tr><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th></tr><tr><td>1</td><td>50</td><td>6</td><td>300</td><td>11</td><td>3600</td></tr><tr><td>2</td><td>75</td><td>7</td><td>600</td><td>12</td><td>4800</td></tr><tr><td>3</td><td>110 (109.92)</td><td>8</td><td>1200</td><td>13</td><td>7200</td></tr><tr><td>4</td><td>135 (134.58)</td><td>9</td><td>1800</td><td>14</td><td>9600</td></tr><tr><td>5</td><td>150</td><td>10</td><td>2400</td><td>15</td><td>1920</td></tr></table>	nn	Rate	nn	Rate	nn	Rate	1	50	6	300	11	3600	2	75	7	600	12	4800	3	110 (109.92)	8	1200	13	7200	4	135 (134.58)	9	1800	14	9600	5	150	10	2400	15	1920
nn	Rate	nn	Rate	nn	Rate																																
1	50	6	300	11	3600																																
2	75	7	600	12	4800																																
3	110 (109.92)	8	1200	13	7200																																
4	135 (134.58)	9	1800	14	9600																																
5	150	10	2400	15	1920																																

**Table 7-2 (continued)**  
Printer port commands

Command	Description																														
C*	When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.																														
nD	Sets data format to values corresponding to n: <table><tr><th>n</th><th>Data bits</th><th>Stop bits</th><th>n</th><th>Data bits</th><th>Stop bits</th></tr><tr><td>0</td><td>8</td><td>1</td><td>4</td><td>8</td><td>2</td></tr><tr><td>1</td><td>7</td><td>1</td><td>5</td><td>7</td><td>2</td></tr><tr><td>2</td><td>6</td><td>1</td><td>6</td><td>6</td><td>2</td></tr><tr><td>3</td><td>5</td><td>1</td><td>7</td><td>5</td><td>2</td></tr></table>	n	Data bits	Stop bits	n	Data bits	Stop bits	0	8	1	4	8	2	1	7	1	5	7	2	2	6	1	6	6	2	3	5	1	7	5	2
n	Data bits	Stop bits	n	Data bits	Stop bits																										
0	8	1	4	8	2																										
1	7	1	5	7	2																										
2	6	1	6	6	2																										
3	5	1	7	5	2																										
F*	When this command is enabled, your Apple IIc accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.																														
I	Echoes printer output on the screen.																														
K	Disables automatic line feed after carriage return.																														
L*	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.																														
M*	When this command is enabled, all incoming line feed characters are masked (removed from the data stream). Normally this command is enabled.																														

**Table 7-2 (continued)**  
Printer port commands

Command	Description																				
nnnN	Changes line width to nnn (from 1 through 255; nnn is optional); does not echo printer output on the screen. <i>Note:</i> 0N does not disable automatic generation of carriage return; to do so, use Z command, put 0 directly in location \$0579, or use the <i>System Utilities</i> disk.																				
nP	Sets parity corresponding to n: <table><tr><th>n</th><th>Parity</th><th>n</th><th>Parity</th></tr><tr><td>0</td><td>None</td><td>4</td><td>None</td></tr><tr><td>1</td><td>Odd</td><td>5</td><td>MARK (1)</td></tr><tr><td>2</td><td>None</td><td>6</td><td>None</td></tr><tr><td>3</td><td>Even</td><td>7</td><td>SPACE (0)</td></tr></table>	n	Parity	n	Parity	0	None	4	None	1	Odd	5	MARK (1)	2	None	6	None	3	Even	7	SPACE (0)
n	Parity	n	Parity																		
0	None	4	None																		
1	Odd	5	MARK (1)																		
2	None	6	None																		
3	Even	7	SPACE (0)																		
R	Resets port 1 and exits from serial port 1 firmware.																				
S	Sends a 233-millisecond BREAK character (used with some printers to synchronize with serial ports).																				
X*	When enabled, this command turns on the XON/XOFF protocol: the Apple IIc looks for the XOFF (\$13) character and responds by halting transmission until an XON (\$11) is received. Normally this command is disabled.																				
Z	Zaps (ignores) further command characters until Control-Reset or PR#1. Does not format output or insert carriage returns into output stream.																				

*Note:* The commands themselves are letter commands, not control characters.

- \* Command (with the exception of L) is available only on the version of the Apple IIc that supports UniDisk 3.5. Command can be toggled: If you follow the command with E (with no intervening space), the command is enabled. If you follow the command with D (with no intervening space), command is disabled. The L command is available on the earlier Apple IIc, but cannot be toggled there.

The serial port 1 command character is set as Control-I when the Apple IIc is turned on. You can change it to a different control character by sending the current control character followed immediately by the new control character you want. This is useful if you want to be able to send Control-I to the printer without firmware intervention. For example, to change the command character from Control-I to Control-V, send Control-I Control-V either from the keyboard or from a program. (Control-V and Control-W are the recommended substitute control characters.) To change the command character back again, send Control-V Control-I. Don't slip any spaces between the control characters that you send.

---

**Warning** Do not use Control-A, -B, -C, -H, -J, -L, -M, or -Y: Apple IIc firmware may intercept these control characters, causing unpredictable results.

---

The following are examples of valid commands and command sequences. These examples all show commands being entered from the keyboard, but your programs can send the characters just as well. Remember to issue a PR#1 before starting to send commands to serial port 1.

To echo output to the display screen:

Control-I I

To set line width 72, disable line feed, and echo:

Control-I K Control-I 7 2 N

To change control character to Control-V:

Control-I Control-V Return

To set up the serial port to allow sending Control-I as part of a character stream:

Control-V (command) Return

---

---

## Characteristics of port 1 at startup

After power-up, the printer firmware sets the following configuration:

- ☐ 9600 baud
- ☐ eight data bits, no parity bits, two stop bits
- ☐ 80-column line width; no echo to display screen
- ☐ firmware supplies line feed after carriage return
- ☐ command character is set to Control-I (see below)

These values are stored in the auxiliary memory screen holes (Table 7-5). You can change some of these settings from the keyboard by typing PR#1, the command character, and one of the commands listed in Table 7-2. How port characteristics change as a result of various activities is described under “Changing Port 1 Characteristics” later in this chapter.

ACIA stands for *asynchronous communication interface adapter*, a serial I/O chip. Note in Chapter 11 that some of the bit assignments for this port differ from those for port 2.

---

---

## Hardware page locations for port 1

Table 7-3 lists for serial port 1 the addresses and bit assignments of its hardware registers on page \$C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

**Warning** This table is for your information only. To avoid having problems with the system, you should **never** try to directly access the hardware. Instead, use the Apple IIc's built-in firmware in your programs.

**Table 7-3**  
Port 1 hardware page locations

Location	Description
\$C090–\$C097	Reserved
\$C098	ACIA transmit/receive data register
\$C099	ACIA status register
\$C09A	ACIA command register
\$C09B	ACIA control register
\$C09C–\$C09F	Reserved

---

---

## I/O firmware support for port 1

Table 7-4 lists the locations and values of the I/O firmware protocol table. This standardized protocol is available for use by any application program. Chapter 3 describes how to use this protocol.

**Table 7-4**  
Port 1 I/O firmware protocol

Address	Value	Description
\$C105	\$38	Pascal ID byte.
\$C107	\$18	Pascal ID byte.
\$C10B	\$01	Generic signature byte of firmware cards.
\$C10C	\$31	Same ID as for Super Serial Card.
\$C10D	\$ii	\$C1ii is entry point of initialization routine (PInit).
\$C10E	\$rr	\$C1rr is entry point of read routine (PRead).
\$C10F	\$ww	\$C1ww is entry point of write routine (PWrite).
\$C110	\$ss	\$C1ss is entry point of the status routine (PStatus).
\$C111	non-zero	No optional routines.

The ACIA register bits are defined in Chapter 11.

---

---

## Screen hole locations for port 1

Table 7-5 lists the screen hole locations that serial port 1 uses. Note that the auxiliary memory locations are reserved for startup value settings, which are listed and interpreted in the table.

**Table 7-5**  
Port 1 screen hole locations

Auxiliary memory screen holes (firmware loads values at power-up)	
Location	Description
\$0478	\$9E (ACIA control reg: eight data + two stop bits, 9600 baud)
\$0479	\$0B (ACIA command reg: no parity)
\$047A	\$40 (flags: no echo, auto LF after CR, serial port)
Bit	Interpretation
7	Echo output on display (0 = no echo)
6	Generate LF after CR (0 = no LF)
5-1	Always = 0 (reserved)
0	1 = communication port; 0 = serial printer port

**Table 7-5 (continued)**  
Port 1 screen hole locations

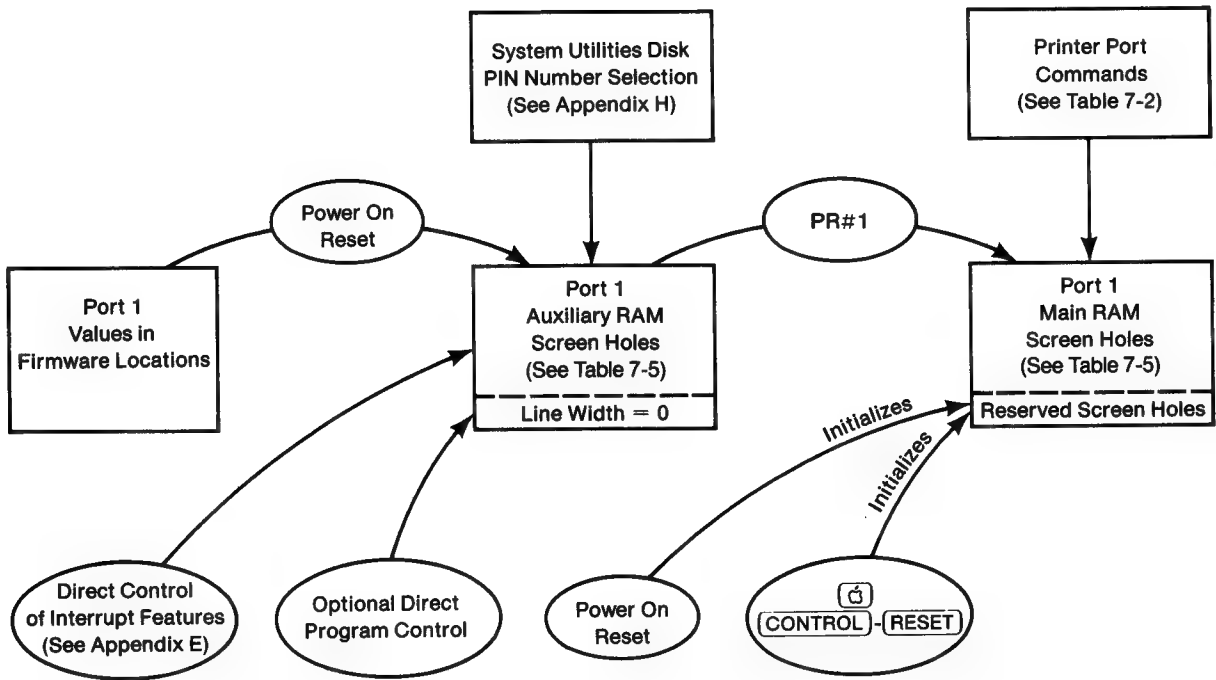
Auxiliary memory screen holes (firmware loads values at power-up)	
Location	Description
\$047B	\$50 (printer width: 80 columns)
<b>Bit</b>	<b>Interpretation</b>
7-0	Printer width (0 = do not insert CR)
Main memory screen holes	
Location	Description
\$0479	Reserved
\$04F9	Reserved
\$0579	Printer width (1-255; 0 = disable formatting)
\$05F9	Temporary storage location
\$0679	Bit 7 = 1 while the firmware is parsing a command string
\$06F9	Current command character (initially Control-I)
\$0779	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return
\$07F9	Current printer column

## Changing port 1 characteristics

Figure 7-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:

- ☐ When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed earlier in this chapter from ROM into the auxiliary memory screen holes listed in Table 7-5.
- ☐ If you specify new characteristics using the *System Utilities* disk, the SUD software changes the values in the auxiliary memory screen holes. Your programs can do the same thing.

- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either Open Apple-Control-Reset or a simple Control-Reset. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up. (See Figure 7-1.)
- PR#1 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location \$0579.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 7-2 to change them.



**Figure 7-1**  
Diagram of port 1 characteristics storage

## Data format and baud rate

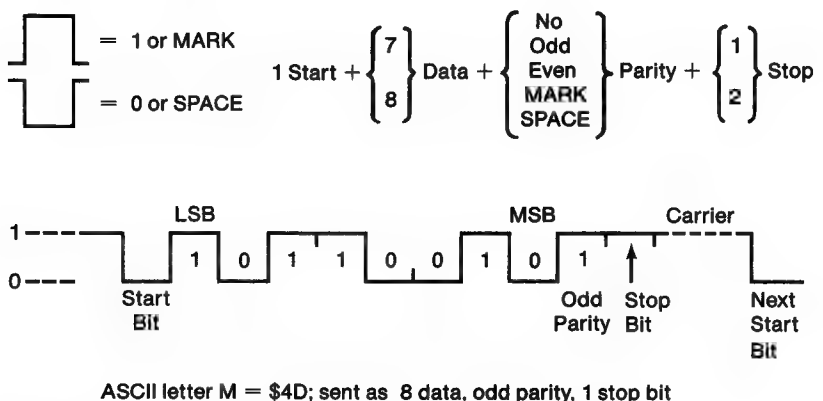
Serial data transfer consists of a string of 1's and 0's sent down a wire at a prearranged rate of transmission, called the baud rate.

Before transfer begins, both sender and receiver look for a continuous value of 1: this is called the carrier (Figure 7-2). When the value goes to 0, the receiver presumes it is a start bit—that is, the bit that designates the beginning of a character of data. If it lasts longer than a bit could possibly last, it is considered a BREAK signal, which some printers use for synchronization.

If the first 0 proves to be a bit, it is interpreted as the start bit. Next come the seven or eight data bits (six is seldom used with computers), low-order bit first. If parity is on, it comes next in the message. Finally, one or two stop bits appear. The stop bits have a value of 1, like the carrier. The next start bit begins transfer of the next character of data.

The parity bit provides a simple check of data validity. Odd parity means the sender counts the number of 1's among the data bits, and sends the appropriate parity bit to make the total number of 1's odd. With even parity, the sender adds the appropriate parity bit to make the total number of 1 bits even. MARK parity is always a 1 bit; SPACE parity is always a 0. The receiver can then check that the parity bit is correct.

If the baud rate is 300 and the data format is one start bit plus seven data bits plus one parity bit plus one stop bit (totaling ten bits transmitted for each byte of data sent), then the actual transfer rate is about 30 characters per second.



**Figure 7-2**  
Data format

---

## Carriage return and line feed

If you are using a typewriter and you push the carriage all the way to the right (in other words, position the printing mechanism at the left margin), you have performed a carriage return. On the other hand, turning the platen so the paper moves to the next line (or using the index key on an electric typewriter) is called a **line feed**. Most typewriters perform a line feed automatically after a carriage return, and so the two seem to be one—but they are not.

Carriage return and line feed are separate ASCII codes. Carriage return is sometimes denoted *CR*; it is ASCII code 13 (\$0D). Line feed, sometimes denoted *LF*, is ASCII code 10 (\$0A). Down Arrow on the Apple IIc keyboard generates a LF.

Some printers can supply a line feed automatically after detecting a carriage return; others cannot. If the printer does not supply a line feed after a carriage return and it is not supplied in the data stream, the printer keeps printing over and over on the same line. On the other hand, if both the printer and the Apple IIc firmware supply LF after CR, double line-spacing results.

If the print head keeps moving too far to the right across the page and then prints many characters on top of one another on the right, then the firmware should be instructed to furnish CR after a certain line width has been reached. If the printer prints too short a line before moving to the next line, then probably the firmware is using too small a line width.

If the printer misses characters at the beginning of each line but otherwise prints correctly, there is probably not enough time for the print mechanism to return to the left margin in response to CR. You must use a lower baud rate with such a printer.

---

## **Sending special characters**

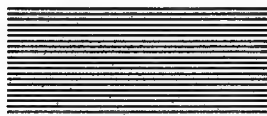
If you want to send special characters (control characters) to the printer without having them intercepted and executed by the Apple IIc firmware, use the Z command (see Table 7-2). If the only special character that causes a problem is the command character (normally Control-I for port 1), you can change just the command character instead of using the zap (Z) command. If you use the zap command, the firmware does no formatting; that is, it does not check line width or insert carriage returns or line feeds. This may be necessary to send graphics to a printer or plotter.

---

## **Displaying output on the screen**

You can display printer output on the screen, but if the printer line width exceeds the 40 or 80 columns you have selected for display, you should turn off video display.





## **Chapter 8**



### **Serial I/O Port 2**

Serial port 2 is one of two serial I/O ports available on the Apple IIc. It is intended primarily as a communication port for modems. You can change it to a serial printer port (like port 1) using the *System Utilities* disk or from a program.

---

**Warning** Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix F for a summary of these differences.

---

Table 8-1 summarizes the characteristics of this port and is a guide to the other information in this chapter. If you change port 2 to a serial printer port, refer to the descriptions in Chapter 7 and use 2 instead of 1 for the port number when required.

The serial port connectors are described in Chapter 11.

**Table 8-1**  
Serial port 2 characteristics

---

<b>Port number</b>	Serial port 2.
<b>Commands</b>	<p>Keyboard commands:</p> <p>IN#2 before Table 8-2 commands, IN#2 to accept port 2 input, PR#1 to echo input to printer, PR#2 to echo input back to port 2.</p> <p>BASIC commands: same.</p> <p>Monitor command: 2 Control-P (does not work if there is an operating system in RAM).</p> <p>All other commands: see Table 8-2.</p>
<b>Initial characteristics</b>	See "Characteristics of Port 2 at Startup."
<b>Hardware page locations</b>	See Table 8-3.
<b>Monitor firmware routines</b>	None.
<b>I/O firmware entry points</b>	See Table 8-4.

**Table 8-1 (continued)**  
Serial port 2 characteristics

---

<b>Use of screen holes</b>	See Table 8-5.
<b>Use of other pages</b>	In terminal mode, firmware uses auxiliary memory locations \$0800–\$087F to store keyboard input, and \$0880–\$08FF as a serial input buffer.

---

---

---

## Using serial port 2

You can access the firmware from BASIC in the usual way—that is, by issuing Control-D (if DOS or ProDOS is in RAM) followed by IN#2 or PR#2. Subsequent input and output are routed through the modem (or other device) connected to serial port 2.

---

<b>Important</b>	In terminal mode, the modem port commands listed in Table 8-2 must follow Control-D and IN#2 ( <i>not</i> PR#2) and the command character (which is usually Control-A).
------------------	---

---

To transfer files to the modem under Pascal, specify REMOUT: or #8:. To transfer files from the modem under Pascal, specify REMIN: or #7:.

Refer to Table 8-4 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

Table 8-2 lists the commands you can use with serial port 2, either from a program or from the keyboard, after you issue IN#2.

---

<b>UniDisk 3.5</b>	Commands followed by an asterisk in Table 8-2 (with the exception of L) are available only on the version of the Apple IIc that supports UniDisk 3.5. These commands can be toggled by following them directly with E (enable) or D (disable).
--------------------	--

---

Each command must be preceded by Control-A (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. If you press Return, you get the current video cursor again. You do not have to press Return (or send a return character) after commands. You can type more than one command on a line, but each must be preceded by the command character.

**Table 8-2**  
Modem port commands

Command	Description																																				
nnn	Sets new line width of nnn (from 1 through 255); this must be followed immediately by N (see below) or by carriage return.																																				
nnB	Sets baud rate to value corresponding to nn: <table><tr><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th></tr><tr><td>1</td><td>50</td><td>6</td><td>300</td><td>11</td><td>3600</td></tr><tr><td>2</td><td>75</td><td>7</td><td>600</td><td>12</td><td>4800</td></tr><tr><td>3</td><td>110 (109.92)</td><td>8</td><td>1200</td><td>13</td><td>7200</td></tr><tr><td>4</td><td>135 (134.58)</td><td>9</td><td>1800</td><td>14</td><td>9600</td></tr><tr><td>5</td><td>150</td><td>10</td><td>2400</td><td>15</td><td>19200</td></tr></table>	nn	Rate	nn	Rate	nn	Rate	1	50	6	300	11	3600	2	75	7	600	12	4800	3	110 (109.92)	8	1200	13	7200	4	135 (134.58)	9	1800	14	9600	5	150	10	2400	15	19200
nn	Rate	nn	Rate	nn	Rate																																
1	50	6	300	11	3600																																
2	75	7	600	12	4800																																
3	110 (109.92)	8	1200	13	7200																																
4	135 (134.58)	9	1800	14	9600																																
5	150	10	2400	15	19200																																
C*	When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.																																				
nD	Sets data format to values corresponding to n: <table><tr><th>n</th><th>Data bits</th><th>Stop bits</th><th>n</th><th>Data bits</th><th>Stop bits</th></tr><tr><td>0</td><td>8</td><td>1</td><td>4</td><td>8</td><td>2</td></tr><tr><td>1</td><td>7</td><td>1</td><td>5</td><td>7</td><td>2</td></tr><tr><td>2</td><td>6</td><td>1</td><td>6</td><td>6</td><td>2</td></tr><tr><td>3</td><td>5</td><td>1</td><td>7</td><td>5</td><td>2</td></tr></table>	n	Data bits	Stop bits	n	Data bits	Stop bits	0	8	1	4	8	2	1	7	1	5	7	2	2	6	1	6	6	2	3	5	1	7	5	2						
n	Data bits	Stop bits	n	Data bits	Stop bits																																
0	8	1	4	8	2																																
1	7	1	5	7	2																																
2	6	1	6	6	2																																
3	5	1	7	5	2																																
F*	When this command is enabled, your Apple IIc accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.																																				
I	Echoes output on the screen.																																				
K	Disables automatic line feed after carriage return.																																				
L*	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.																																				

**Table 8-2 (continued)**  
Modem port commands

Command	Description																				
M*	When this command is enabled, all incoming line feed characters are masked (removed from the data stream). Normally this command is enabled.																				
nnnN	Sets line width to nnn (from 1 through 255); does not echo output on the screen. <i>Note:</i> 0N does not disable automatic generation of carriage return; to do so, use the Z command, put 0 directly in location \$057A, or use the <i>System Utilities</i> disk.																				
nP	Sets parity corresponding to n: <table><tr><th>n</th><th>Parity</th><th>n</th><th>Parity</th></tr><tr><td>0</td><td>none</td><td>4</td><td>none</td></tr><tr><td>1</td><td>odd</td><td>5</td><td>MARK (1)</td></tr><tr><td>2</td><td>none</td><td>6</td><td>none</td></tr><tr><td>3</td><td>even</td><td>7</td><td>SPACE (0)</td></tr></table>	n	Parity	n	Parity	0	none	4	none	1	odd	5	MARK (1)	2	none	6	none	3	even	7	SPACE (0)
n	Parity	n	Parity																		
0	none	4	none																		
1	odd	5	MARK (1)																		
2	none	6	none																		
3	even	7	SPACE (0)																		
Q	Quits terminal mode.																				
R	Resets port 2 and exits from serial port 2 firmware.																				
S	Sends a 233-millisecond BREAK character.																				
T	Enters terminal mode. Use this command after IN#2 only. Also, if you follow this command by PR#2, the Apple IIc echoes input to output. (If the other device does so too, the first character loops endlessly, locking up the system. Use Control-Reset to get out.)																				
X*	When enabled, this command turns on the XON/XOFF protocol: the Apple IIc looks for the XOFF (\$13) character and responds by halting transmission until an XON (\$11) is received. Normally this command is disabled.																				
Z	Zaps (ignores) further command characters until Control-Reset. Does not format output or insert carriage returns into output stream.																				
Control-T	This command from a remote device puts the Apple IIc in terminal mode if IN#2 is already in effect. It is the same as Control-A T typed locally.																				

**Table 8-2 (continued)**  
Modem port commands

Command	Description
Control-R	This command from a remote device undoes the terminal mode command. If IN#2 and PR#2 are in effect, the remote keyboard and display become the input and output devices of the local Apple IIc. It is the same as Control-A Q typed locally.

*Note:* The commands themselves are letter commands, not control characters.

- Command (with the exception of L) available only on the version of the Apple IIc that supports UniDisk 3.5. Command can be toggled: If you follow the command with E (with no intervening space) the command is enabled. If you follow the command with D (with no intervening space) the command is disabled. The L command is available on the earlier Apple IIc, but it cannot be toggled there.

When the Apple IIc is turned on, the serial port 2 command character is defined as a Control-A. You can change it to a different control character by typing the current control character followed immediately by the new control character you want. This is useful if you want to be able to send Control-A to the output device without firmware intervention.

For example, to change the command character from Control-A to Control-V, send Control-A Control-V either from the keyboard or from a program. (Control-V and Control-W are the recommended substitute control characters.) To change the command character back again, send Control-V Control-A.

---

**Warning** Do not use Control-B, -C, -H, -I, -J, -L, -M, or -Y: Apple IIc firmware may intercept these control characters, causing unpredictable results.

---

The following are examples of valid commands and command sequences. These examples show commands being entered from the keyboard, but your programs can send the characters just as well.

To enable echo to the screen:

Control-A I

To send a break character to a remote device:

Control-A B

To change the control character to Control-V (for example, so you can send Control-A as part of a character stream):

Control-A Control-V Control-V(command)

---

---

## Characteristics of port 2 at startup

After power-up, the firmware sets the following configuration:

- ☐ 300 baud
- ☐ eight data bits, no parity bits, one stop bit
- ☐ firmware does not supply line feed after carriage return
- ☐ firmware does not insert carriage returns into output stream
- ☐ firmware does not echo output to the display screen
- ☐ command character is set to Control-A

These values are stored in the auxiliary memory screen holes (Table 8-5). You can change some of these settings from the keyboard using the command character followed by one of the commands listed in Table 8-2. How port characteristics change as a result of various activities is described later in this chapter.

If you change any of these values using keyboard commands or commands from a program, subsequent accesses to the port firmware (even by another program) use the new settings instead of the power-up values. This allows you to change the settings once at system startup and get the desired configuration for subsequent uses.

---

---

## Hardware page locations for port 2

Table 8-3 lists for serial port 2 the addresses of its hardware registers on page \$C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

---

<b>Warning</b>	This table is for your information only. To avoid having problems with your system, you should <b>never</b> try to directly access the hardware. Instead, use the Apple IIc's built-in firmware in your programs.
----------------	---

---

**Table 8-3**  
Port 2 hardware page locations

Location	Description
\$C0A0-\$C0A7	Reserved
\$C0A8	ACIA transmit/receive data register
\$C0A9	ACIA status register
\$C0AA	ACIA command register
\$C0AB	ACIA control register
\$C0AC-\$C0AF	Reserved

Note in Chapter 11 that some of the bit assignments for this port differ from those for port 1.

## I/O firmware support for port 2

Table 8-4 lists the values in the I/O firmware protocol table for serial port 2. This standardized protocol is available for use by any application program. Chapter 3 describes how to use this protocol.

**Table 8-4**  
Port 2 I/O firmware protocol

Address	Value	Description
\$C205	\$38	Pascal ID byte.
\$C207	\$18	Pascal ID byte.
\$C20B	\$01	Generic signature byte of firmware cards.
\$C20C	\$31	Same ID as for Super Serial Card.
\$C20D	\$ii	\$C2ii is entry point of initialization routine (PInit).
\$C20E	\$rr	\$C2rr is entry point of read routine (PRead).
\$C20F	\$ww	\$C2ww is entry point of write routine (PWrite).
\$C210	\$ss	\$C2ss is entry point of the status routine (PStatus).
\$C211	non-zero	No optional routines.

## Screen hole locations for port 2

Table 8-5 lists the screen hole locations that serial port 2 uses. Note that the auxiliary memory locations are reserved for startup value settings, which are listed and interpreted in the table.

The ACIA register bits are defined in Chapter 11.

**Table 8-5**  
Port 2 screen hole locations

---

**Auxiliary memory screen holes (firmware loads values at power-up)**

---

Location	Description										
\$047C	\$16 (ACIA control reg: eight data + one stop bit, 300 baud)										
\$047D	\$0B (ACIA command reg: no parity)										
\$047E	\$01 (flags: no echo, no auto LF after CR, communication port)										
	<table> <tr> <th>Bit</th><th>Interpretation</th></tr> <tr> <td>7</td><td>Echo output on display (0 = no echo)</td></tr> <tr> <td>6</td><td>Generate LF after CR (0 = no LF)</td></tr> <tr> <td>5-1</td><td>Always = 0 (reserved)</td></tr> <tr> <td>0</td><td>1 = communication port; 0 = serial printer port</td></tr> </table>	Bit	Interpretation	7	Echo output on display (0 = no echo)	6	Generate LF after CR (0 = no LF)	5-1	Always = 0 (reserved)	0	1 = communication port; 0 = serial printer port
Bit	Interpretation										
7	Echo output on display (0 = no echo)										
6	Generate LF after CR (0 = no LF)										
5-1	Always = 0 (reserved)										
0	1 = communication port; 0 = serial printer port										
\$047F	\$00 (line length: do not add any CR to output stream)										
	<table> <tr> <th>Bit</th><th>Interpretation</th></tr> <tr> <td>7-0</td><td>Line length (0 = do not insert CR)</td></tr> </table>	Bit	Interpretation	7-0	Line length (0 = do not insert CR)						
Bit	Interpretation										
7-0	Line length (0 = do not insert CR)										

---

**Main memory screen holes**

---

Location	Description
\$047A	Reserved
\$04FA	Reserved
\$057A	Line length (1-255; 0 = disable formatting)
\$05FA	Temporary storage location
\$067A	Bit 7 = 1 if and only if the firmware is currently parsing a command string
\$06FA	Current command character (initially Control-D)
\$077A	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return
\$07FA	Current column

---

---

## Changing port 2 characteristics

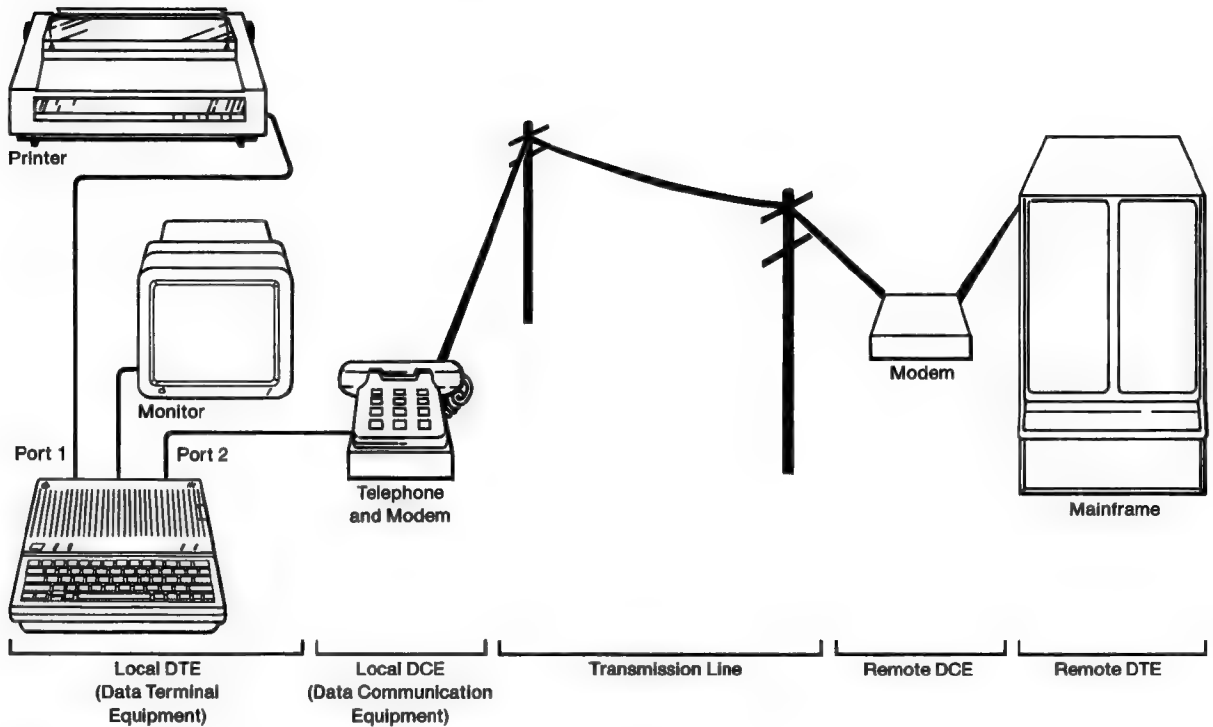
Figure 8-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:

- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Table 8-2 from ROM into the auxiliary memory screen holes listed in Table 8-5.
- If you specify new characteristics using the *System Utilities* disk, the utility software changes the values in the auxiliary memory screen holes.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either Open Apple-Control-Reset or a simple Control-Reset. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.
- IN#2 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location \$057A.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 8-2 to change these characteristics.



- the telephone lines, with their switching equipment, boosters, and noise
- some combination of modem, cable, and remote computer or terminal

As you can imagine, some method is required for successful data transmission. If you have problems, change only one variable at a time and then cycle through the other variables one at a time. Take nothing for granted. The data format advertised for an information service, for example, may be different from the one you end up using with the Apple IIc.



**Figure 8-2**  
Devices in a typical communication setup

---

## Carriage return and line feed

If you are communicating with a computer or terminal, carriage return and line feed may or may not be involved. Start off without generating them, and turn on automatic generation only as needed. They are described as used with printers in Chapter 7.

---

## Routing input and output

This section discusses the possible ways that serial port 2 can route information. Sometimes the cause of communication problems is that information is not going where you think it is, or it is and you cannot see evidence of the fact. Figures 8-3 through 8-6 show some of the patterns of information flow you can select.

It is best to read all this material as a unit; questions that arise while you read one description may be answered elsewhere.

The simplest serial port 2 command is IN#2 (Figure 8-3). Port 2 becomes the input device. Data coming into the port gets passed to the input buffer (page \$02 of main memory). Applesoft firmware and system software can see the data and carry out commands in the normal way.

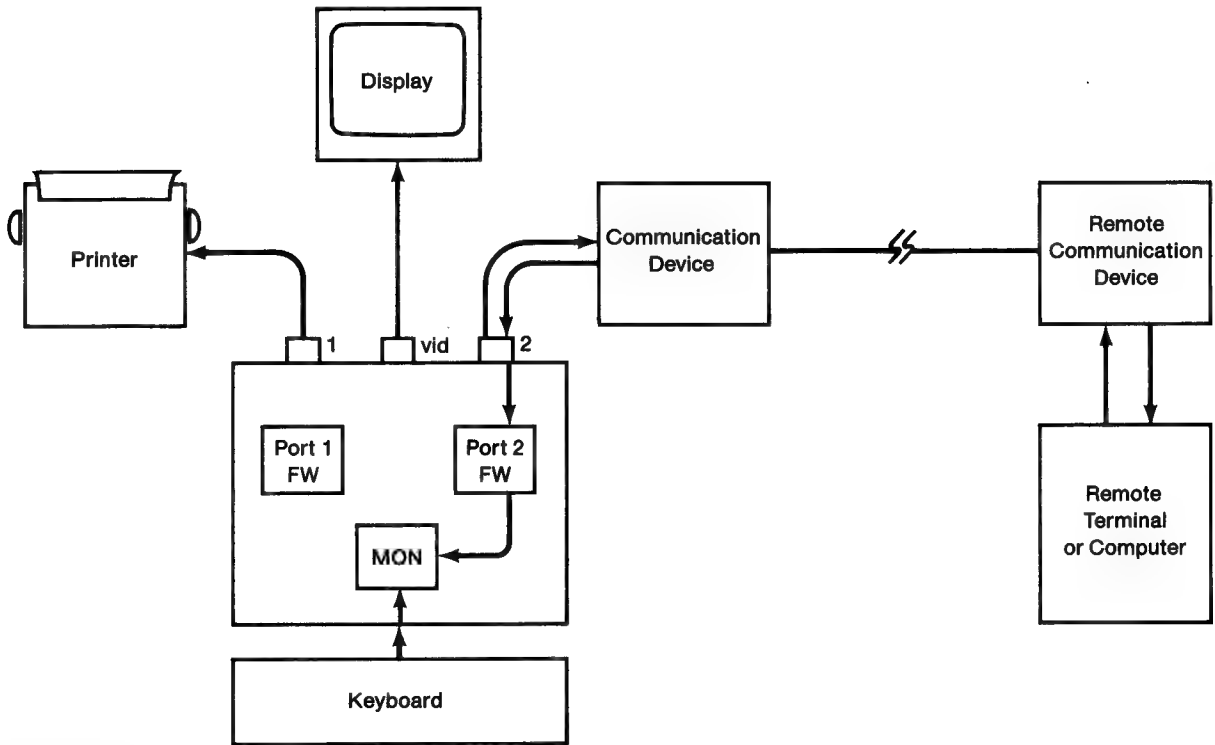
Of course, you can also use just the PR#2 command—for example, if you want to send a listing to the modem.

To use port 2 for data communication, you ordinarily put it into terminal mode. Following IN#2, pressing Control-A gets the attention of the port 2 firmware, which displays a blinking question mark (?) as a prompt. Now type T to put the computer in terminal mode. In this mode, the firmware displays a blinking underscore character ( \_ ) as a prompt.

In the discussion that follows, *local* refers to your Apple IIc. *Remote* refers to some other device, usually in a distant location and at the other end of a communication link. The remote device can be any ASCII-generating unit: a terminal or a computer.

If a remote computer is another Apple IIc or an Apple II series machine with a Super Serial Card in it, then *most* of the commands described here apply to it as well.

For a further description of what terminal mode does and how to get into and out of it, refer to the last section of this chapter.

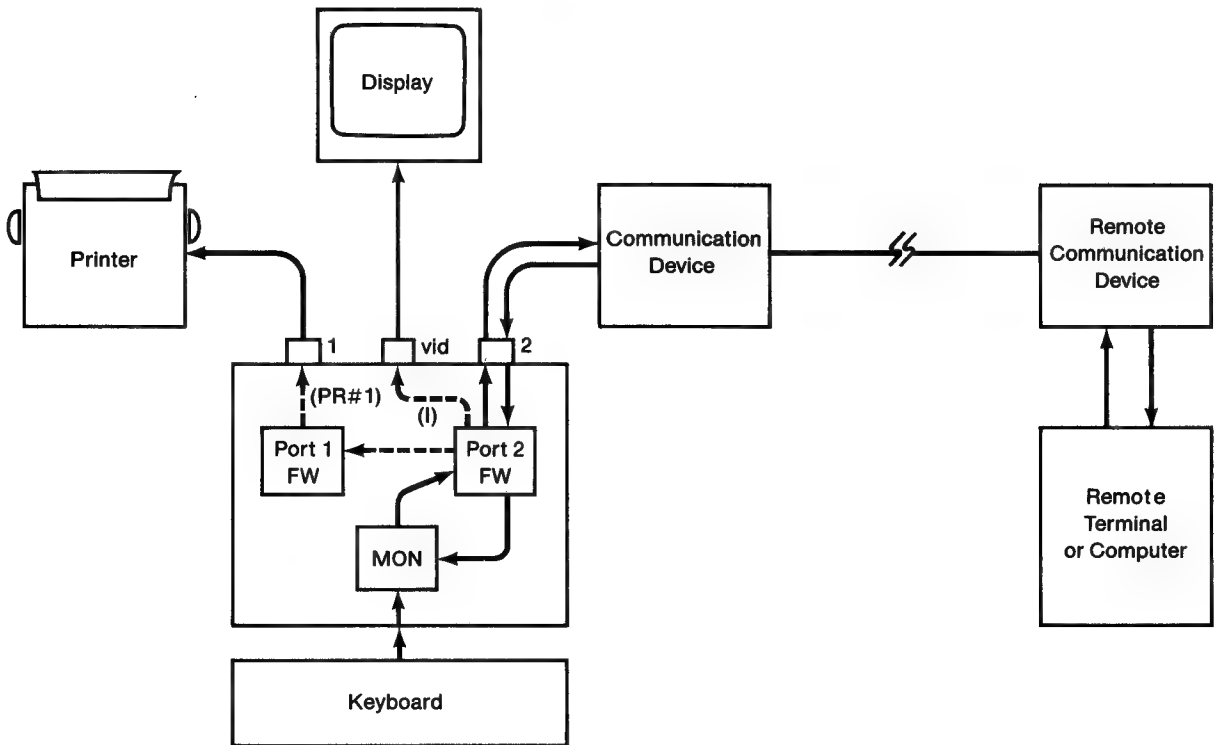


**Figure 8-3**  
Effect of IN#2

### Half-duplex operation

In half-duplex operation, information can flow from A to B or from B to A, but in only one direction at a time. In a half-duplex setup, the host does not echo back to the terminal what the terminal sends it. For half-duplex operation, use IN#2 and Control-A T (Figure 8-4) whether the Apple IIc is the host or the terminal.

IN#2 plus Control-A T is the best way to set up the computer for auto-answer operation. The T command allows port 2 firmware to exchange information with the local modem without interference from the local firmware or system software. (The remote device can always cancel the T command with Control-R if necessary, and restore terminal mode with Control-T.) Avoiding PR#2 at this point means that the Apple IIc can operate as a half-duplex terminal, half-duplex host, or full-duplex terminal. (The remote device can also issue Control-A PR#2 if PR#2 is required at the local computer.)



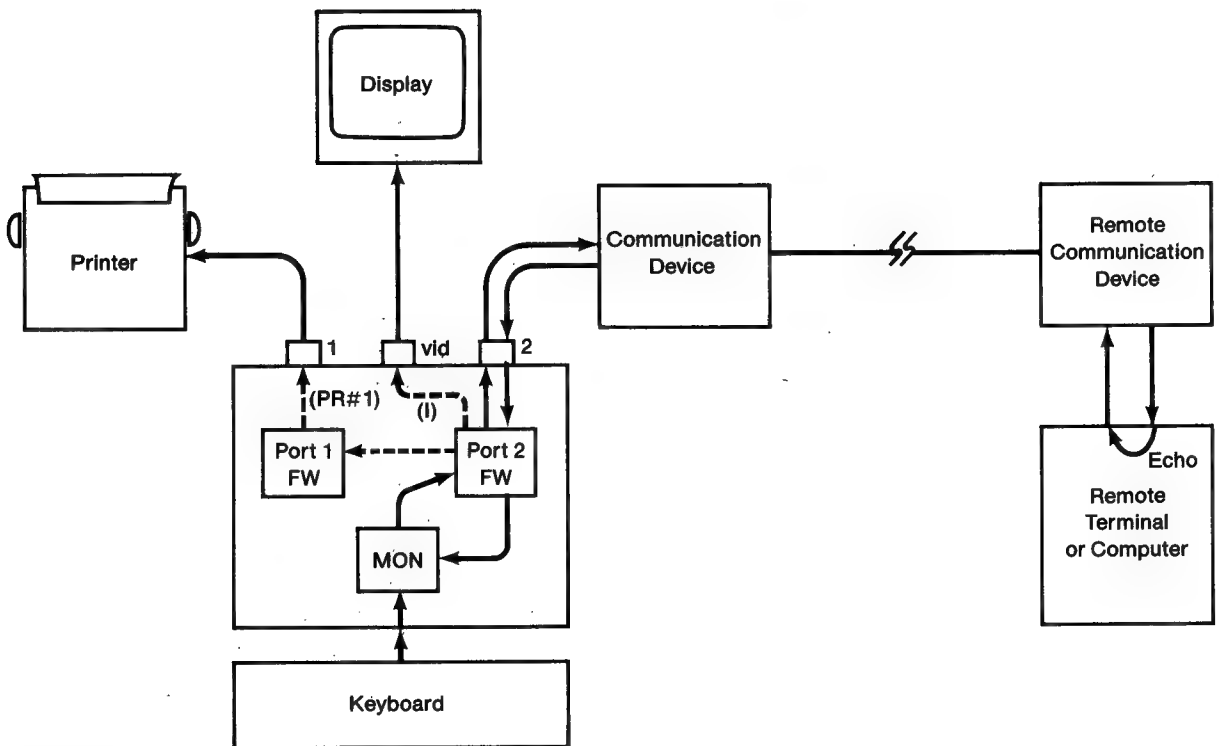
**Figure 8-4**  
Effect of IN#2 and T command, half duplex

In half-duplex operation, the output hook is available for other uses. For example, you can issue PR#1 to print incoming messages from port 2. Use the Control-A I command to display information on the screen.

## Full-duplex operation

In full-duplex operation, information can flow from A to B and from B to A simultaneously. Typically, one of the computers (the host computer) echoes its input to output, so the other computer (the terminal) can easily verify that the communication is taking place.

Figure 8-5 shows the flow of information when the Apple IIc is a full-duplex terminal. (The setup commands, IN#2 and Control-A T, are the same as for half duplex.)

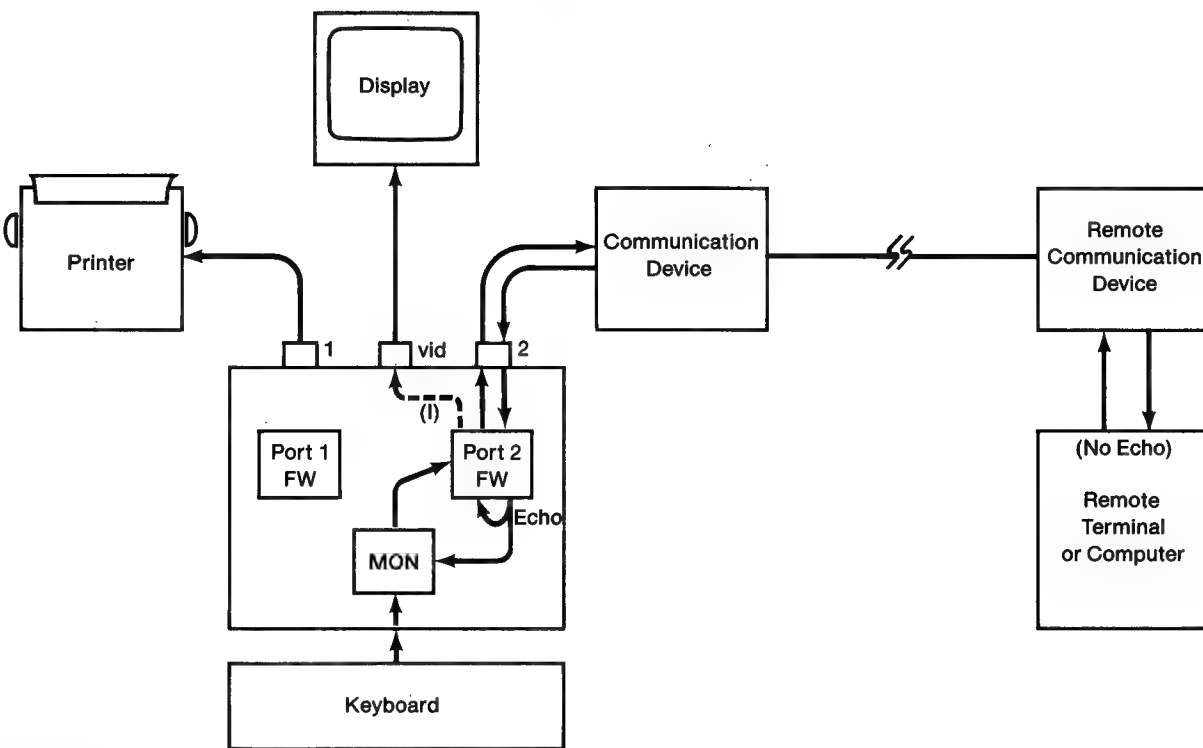


**Figure 8-5**  
Effect of IN#2 and T command, full-duplex terminal

If your Apple IIc is the terminal in full-duplex operation, use the N command to turn off echoing input to the screen. If the Apple IIc does echo input to the screen in this setup, everything you type will appear twice: once from the Apple IIc and once from the host computer.

In this mode of operation, if you echo input to the printer you can get a printed record of both sides of the communication session: the input from the host, and the Apple IIc output as echoed by the host.

Figure 8-6 shows the flow of information when the Apple IIc is a full-duplex host. In this case, the local Apple IIc must echo input to output for the remote device. The setup commands include PR#2 in this case.



**Figure 8-6**  
Effect of IN#2, PR#2, and T command, full-duplex host

**Warning** If the Apple IIc echoes input to output and the other computer does too, then the first subsequent keypress will echo back and forth endlessly and lock up the Apple IIc. This will require a Control-Reset to get out.

If you echo input to output when using an Information service, the host will end up seeing the echo of what it sent you as though you had typed it.

In this arrangement, the local output hook is not available for using the printer or other device. To display keyboard and port 2 input on the screen, issue Control-A I.

## Terminal mode

Terminal mode makes the Apple IIc act like a dumb terminal—one that just sends and receives information, but does not process it. Input and output flow through special serial I/O buffers on page \$08 of auxiliary memory. Applesoft firmware and system software cannot see or interpret the data: only the serial port 2 firmware deals with it.

In most terminal mode setups, the firmware will not display port 2 input unless you use the Control-A I command.

---

### Warning

When using terminal mode, \$0800–\$08FF of auxiliary RAM is used for buffering. Any data stored there will be overwritten when terminal mode is enabled.

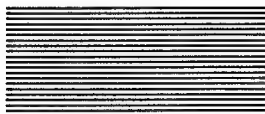
---

Control-A T turns on terminal mode, and Control-A Q turns it off.

The remote device can go into terminal mode, and then turn off the local Apple IIc's terminal mode with the Control-R command. If it then issues Control-A PR#2, local output will go to the remote device. The remote keyboard and display then become the input and output devices of the local Apple IIc processor. This is remote mode.

In remote mode, the local Apple IIc does not use the serial I/O buffers (as it does in terminal mode); therefore, local firmware and system software detect and interpret all input and output data. So, for example, if you type CATALOG at the remote device keyboard, the local Apple IIc will execute the command and list the disk catalog on the remote device's display. (In terminal mode, the local computer would simply display the word CATALOG on its screen.)

The remote device can turn the local Apple IIc's terminal mode back on with Control-T. Control-A T issued at the remote device only turns on the remote device's terminal mode, unless the command character there has already been changed to something else.



## **Chapter 9**



# **Mouse and Game Input**

This chapter describes the Apple IIc's mouse port and hand controller (game) input capabilities. The mouse and hand controllers use the same 9-pin connector on the back panel; the firmware uses the port as directed by keyboard or program commands.

---

---

## Mouse input

Table 9-1 summarizes the mouse port's characteristics and guides you to other information in this part of the chapter.

---

**Warning** If you want your programs that use the mouse on the Apple IIe and other Apple II series computers to work with the Apple IIc, always use the I/O firmware entry points listed in Tables 9-4 and 9-5, rather than dealing directly with the mouse hardware and RAM locations.

---

The mouse back panel connector is described in Chapter 11.

**Table 9-1**  
Mouse Input port characteristics

---

<b>Port number</b>	Mouse input port 4.
<b>BASIC commands</b>	Turn on mouse: <code>PRINT CHR\$(4)"PR#4":PRINT CHR\$(1)</code>  Turn off mouse interrupts: <code>PRINT"PR#4":PRINT CHR\$(0)</code>  Turn on graphics character set: See "MouseText" in Chapter 5.
<b>Initial characteristics</b>	After a reset, all mouse interrupts are off and the rising edge of X0 and Y0 are selected for interrupts.
<b>Hardware page locations</b>	See Table 9-2.
<b>Monitor firmware routines</b>	None.
<b>I/O firmware entry points</b>	See Tables 9-3 and 9-4.
<b>Use of screen holes</b>	See Table 9-5.

## Memory expansion

---

The memory expansion version of the Apple IIc places the mouse at input port 7 and the memory expansion card at port 4. Thus, all "PR4" entries become "PR7" entries.

---

---

## Mouse connector signals

The mouse uses the same DB-9 connector as the hand controllers. However, the interpretation of the signals arriving on the pins differs depending on the commands and signals received.

Figure 11-37 shows the names of the pin assignments when a mouse is connected.

---

## Mouse operating modes

This section tells what the mouse operating modes are for. Later sections of this chapter describe how to set the various mouse operating modes.

Your program should call the `ServeMouse` routine to determine the source of an interrupt as soon as it receives one, in all the interrupt modes except transparent mode.

### Transparent mode

In transparent mode, your program must read screen holes to check for mouse movement. An interrupt routine in the Apple IIc firmware updates mouse position counters each time the mouse is moved, then returns control to the main program task. The findings of the interrupt routine are placed in the screen holes for your program to find. Table 9-5 lists the screen holes with the information that your program should look for.

This is the only mouse mode available to BASIC programs.

### Movement interrupt mode

On the Apple IIc, a signal called *VBIInt* can interrupt the processor whenever a video vertical blanking signal occurs. This can make it easier for your programs to smoothly move the mouse cursor in response to mouse movements.

"MouseText" In Chapter 5 contains recommendations for using MouseText characters with a mouse.

In movement interrupt mode, the mouse firmware arms VBLInt whenever the mouse is moved at least one count in any direction. When VBLInt occurs, program control passes to the vector address contained at locations \$03FE and \$03FF; the interrupt handler in your program can then update the cursor smoothly to its next screen position.

Your program's interrupt handler must first call ServeMouse (Table 9-3) to see if the mouse caused the interrupt. It should then call ReadMouse to get mouse status and its current X-Y position. The routine can also change the mouse mode and position if desired.

The maximum amount of mouse movement that can occur between successive VBLInt interrupts is limited to the distance someone can move a mouse in one-sixtieth of a second.

### **Button interrupt mode**

The Apple IIc mouse-button hardware location does not generate interrupts. However, a program can simulate mouse-button interrupts by polling the button whenever VBLInt occurs, and acting on the interrupt whenever the button state has changed. This lets your program provide fast response to the mouse movement without too much program overhead.

### **Movement/button interrupt mode**

The movement/button interrupt mode is a combination of the two modes just described. It provides the best response possible without constant polling of the mouse position and button. Your program can effectively process a main task concurrently with cursor and menu updating, as well as menu-selected command processing.

### **Vertical blanking active modes**

The vertical blanking active modes are the same as the four just described except that they allow VBL interrupts to be sent to the user.

Appendix E explains how the firmware handles interrupts.

## Mouse soft switches

The soft switches assigned to the mouse interface are shown in Table 9-2. On power-up or reset, the hardware selects the rising edge of X0 and Y0 (mouse movement signals) and masks out all mouse interrupts.

**Warning** Table 9-2 is included here for your information only. You should use the built-in firmware to access the mouse; doing so is much easier than writing your own mouse interrupt handler and guarantees compatibility with all other Apple II-series computers.

**Table 9-2**  
Mouse soft switches

Name	Action	Hex	Function
IOUDis	W	\$C07E	On: Disable IOU access for addresses \$C058 to \$C05F; enable access to DHiRes switch*
IOUDis	W	\$C07F	Off: Enable IOU access for addresses \$C058 to \$C05F; disable access to DHiRes switch*
RdIOUDis	R7	\$C07E	Read IOUDis switch (1 = off)†
DisXY	R/W	\$C058	Disable (mask) X0 and Y0 movement interrupts‡
EnbXY	R/W	\$C059	Enable (allow) X0 and Y0 movement interrupts‡
RdXYMsk	R7	\$C040	Read status of X0/Y0 interrupt mask (1 = mask on)
RstXY	R	\$C048	Reset X0/Y0 interrupt flags
X0Edge	R/W	\$C05	Select rising edge of X0 for interrupt‡
X0Edge	R/W	\$C05D	Select falling edge of X0 for interrupt‡
RdX0Edge	R7	\$C042	Read status of X0 edge selector (1 = falling)
RstXInt	R	\$C015	Reset mouse X0 interrupt flag

**Table 9-2** (continued)  
Mouse soft switches

Name	Action	Hex	Function
Y0Edge	R/W	\$C05E	Select rising edge of Y0 for interrupt†
Y0Edge	R/W	\$C05F	Select falling edge of Y0 for interrupt†
RdY0Edge	R7	\$C043	Read status of Y0 edge selector (1 = falling)
RstYInt	R	\$C017	Reset mouse Y0 interrupt flag
DisVBl	R/W	\$C05A	Disable (mask) VBL interrupts‡
EnVBl	R/W	\$C05B	Enable (allow) VBL interrupts‡
RdVBIMsk	R7	\$C041	Read status of VBL interrupt mask (1 = mask on)
RstVBl	R	\$C019	Read and then reset VBLInt flag
PTrig	R/W	\$C070	Reset VBLInt flag; trigger paddle timer
RdBtn0	R7	\$C061	Read first mouse button status (1 = pressed)§
Rd63		R7	\$C063 Read second mouse button status (0 = pressed)¶
MouX1	R7	\$C066	Read status of X1 (mouse X direction) (1 = high)
MouY1	R7	\$C067	Read status of Y1 (mouse Y direction) (1 = high)

\* When IOUDis is on, \$C058–\$C05F do not affect mouse, and \$C05E and \$C05F become DHiRes (Table 5-8).

† Read or write to \$C07x also resets VBLInt and triggers paddle timers.

‡ These work only if IOUDis is off.

§ This location is also the Open Apple key (Table 4-1).

¶ This is also the location of the Shift-key mod (Appendix F).

Mouse firmware sets interrupts in response to mode settings under program control. The vertical blanking interrupt (VBLInt) is armed if the mouse button is pushed or moves at least a count of 1 in the X0 or Y0 coordinates. Read \$C070 to reset the VBL interrupt. Because a VBL occurs every sixtieth of a second, that is the maximum time that can elapse before the resulting interrupt can be acknowledged and acted upon.

Software can also select which edge of X0 and Y0 information will cause the XInt or YInt.

When an interrupt has occurred, you can read the direction of the mouse's X1 movement by reading address \$C066 bit 7, and Y1 movement by reading address \$C067 bit 7.

A program can read the status of the soft switches by reading one of the locations \$C040–\$C043 and then testing data bit 7. The soft switches are described in Table 9-2.

The section on mouse input in Chapter 11 explains what X0, Y0, X1, Y1 are and what they mean with respect to mouse movement.

If you write your own mouse interrupt handler, it should enable the main bank-switched memory, set up its own IRQ vectors at addresses \$FFFE and \$FFFF, keep track of video modes and the alternate stack, and check for the interrupt source in the same manner as the mouse firmware listed in Appendix I, beginning at address \$C400.

---

<b>Important</b>	The listing in Appendix I provides source code only for the memory expansion version of the Apple IIc. Mouse code starts at \$C700 in the new ROM. There are instructions for obtaining listings for the original and UniDisk 3.5 versions in Appendix I.
------------------	---

---

---

<b>UniDisk 3.5</b>	The 32K ROM includes a new feature for programs that need to use mouse interrupts for their own purposes. If your program sets bit 7 of the mouse port mode byte at \$07FC (\$C7FF in the memory expansion IIc) to 1, mouse movement interrupts will be passed to the interrupt handler of your program. VBL interrupts will still be handled by the Apple IIc's firmware. You should use this feature only if the mouse firmware can't keep up with your needs.
--------------------	--

---

---

## I/O firmware support for mouse input

---

<b>Memory expansion</b>	The memory expansion version of the Apple IIc places the mouse at \$C700 and the memory expansion card at \$C400. This means that the mouse is supported on page \$C7 in the new Apple IIc, so change all \$C4 and \$40 addresses to \$C7 and \$70.
-------------------------	---

---

The Apple IIc supports the mouse with firmware starting at address \$C400. This firmware is necessary because the mouse requires fast, transparent interrupt processing to work effectively.

In assembly language you can use direct firmware support for sophisticated mouse applications. To enable the mouse, first load a mode byte into the accumulator (and \$C4 in X, \$40 in Y) and then do a JSR to the firmware routine called *SetMouse* (Table 9-3). Valid mode bytes are the following:

\$00	Turns mouse off
\$01	Sets transparent mode
\$03	Sets movement interrupt mod
\$05	Sets button interrupt mode
\$07	Sets movement or button interrupt mode
\$08	Turns mouse off, VBIInt active
\$09	Sets transparent mode, VBIInt active
\$0B	Sets movement interrupt mode, VBIInt active
\$0D	Sets button interrupt mode, VBIInt active
\$0F	Sets movement or button interrupt mode, VBIInt active

The firmware then initializes the mouse. To read the current position and status of the mouse, first load \$C4 into the X register, load \$40 into the Y register, save processor status, disable interrupts, and then JSR to the firmware routine called *ReadMouse* (Table 9-3), which stores the information in the port 4 screen holes (Table 9-5).

Table 9-3 lists the mouse port firmware routine offsets. Each address contains the low byte of the entry point of the routine described. The calling setup for all routines (except *ServeMouse*) is the same: the X register must contain \$C4, and the Y register must contain \$40. When the routine has finished, the A, X, and Y register contents are undefined.

---

### Memory expansion

The memory expansion version of the Apple IIc places the mouse at \$C700 and the memory expansion card at \$C400. Thus, all mouse firmware routines start at a \$C7XX address, instead of \$C4XX.

---

**Table 9-3**  
Mouse firmware routines

Location	Offset for	Description
\$C412	SetMouse	Sets the mouse mode to the value in the accumulator. Input: A register contains mode (see \$07FC, Table 9-5) (\$07FF in new Apple IIc). Output: Carry bit = 0 means mode was legal; carry bit = 1 means mode was not legal.
\$C413	ServeMouse	Serves mouse interrupt if needed. Input: X, Y, A registers—doesn't matter. Output: Carry bit = 0 means mouse caused the interrupt; carry bit = 1 means something else caused it. This routine updates \$077C (\$077F in new Apple IIc) to show which event caused the interrupt (values in Table 9-5).
\$C414	ReadMouse	Updates screen holes to show current mouse X-Y position and button status; clears VBIInt, button and movement interrupt bits in the status byte. Doesn't reenable interrupts until after retrieving position values. Output: Carry bit = 0.
\$C415	ClearMouse	Sets the mouse position to 0, though not necessarily within clamping boundaries; leaves button and interrupt bits in status byte unchanged. Output: Carry bit = 0.
\$C416	PosMouse	Sets the mouse coordinates to new values. Input: X and Y screen holes contain new X and Y positions. Output: Carry bit = 0.

**Table 9-3 (continued)**  
**Mouse firmware routines**

Location	Offset for	Description
\$C417	ClampMouse	Sets new clamping boundaries (see Table 9-5). Does not affect mouse position or update mouse position screen holes; use ReadMouse to do that. Input: A register = 0 means set new X boundaries; A register = 1 means set new Y boundaries. Output: Carry bit = 0.
\$C418	HomeMouse	Sets the internal mouse position to the upper-left corner of the clamping window. Does not update mouse position screen holes; use ReadMouse to do that.
\$C419	InitMouse	Sets startup internal values; does not update mouse-position screen holes. Output: Carry bit = 0.

Here is a sample sequence of events and calls:

1. Four screen holes contain the mouse's X and Y coordinates, and one contains the status of the last mouse movement (Table 9-5).
2. Call InitMouse.
3. Inhibit interrupts, set up the boundaries you want, then call ClampMouse.
4. Use PosMouse, HomeMouse, or ClearMouse to position the mouse where you want it.
5. Put the mouse mode (see address \$07FC in Table 9-5) that you want to use in the accumulator, then call SetMouse (use address \$07FF for the new Apple IIc).
6. If you have set one of the interrupt modes, then when an interrupt arrives, call ServeMouse to determine the source of the interrupt.
7. Disable interrupts and call ReadMouse. Retrieve the position values, then reenale interrupts.

## Pascal support

Table 9-4 lists the locations and values of the I/O firmware protocol that Pascal 1.1 and later versions use. However, Pascal must use a special attach driver to support the mouse.

### Memory expansion

The memory expansion version of the Apple IIc places the mouse at \$C700 and the memory expansion card at \$C400. Thus, all mouse firmware routines start at a \$C7XX address, instead of \$C4XX.

**Table 9-4**  
Mouse port I/O firmware protocol

Address	Value	Description
\$C405	\$38	Pascal ID byte
\$C407	\$18	Pascal ID byte
\$C40B	\$01	Generic signature byte of firmware cards
\$C40C	\$20	2 = X-Y pointing device; 0 = identification code
\$C40D		Initialization routine (not implemented; returns error code)
\$C40E		Standard read routine (not implemented; returns error code)
\$C40F		Standard write routine (not implemented; returns error code)
\$C410		Standard status routine (not implemented; returns error code)
\$C411	\$00	Optional routines follow
\$C4FB	\$D6	A mouse identification byte

## BASIC and assembly-language support

### Memory expansion

The memory expansion version of the Apple IIc places the mouse at \$C700 and the memory expansion card at \$C400. This means that all "PR4" or "IN4" calls change to "PR7" or "IN7" calls.

In BASIC you must turn the mouse on by printing PR#4 and then CHR\$(1) before you can get input from the mouse. This sets transparent mode. After that, reenables video output with PR#3 and takes subsequent input from the mouse by issuing IN#4. The first input statement after that (INPUT X,Y,S) initializes and enables the mouse and returns a three-element string

```
+xxxx, +yyyy, +st
```

representing the x-coordinate, y-coordinate, and status digits.

The coordinates will be integers between 0 and +1023. These are called the clamping boundaries of the mouse.

The sign preceding the status digits is normally positive; it becomes negative when you press a key on the keyboard.

The first digit, s, of the status is 0. The second digit, t, of the status is 1 if the mouse button is still pressed, 2 if it was just pressed, 3 if it was just released, and 4 if it is still released.

To disable the mouse, use these statements:

```
PRINT CHR$(4) "PR#4"  
PRINT CHR$(0)  
PRINT CHR(4) "PR#3"
```

---

<b>Important</b>	Change all 4's to 7's for the memory expansion version.
------------------	---

---

---

## Screen holes

Table 9-5 lists the screen holes that the mouse firmware uses. Note that the mouse firmware reserves port 5 screen holes for its own use. Also, the auxiliary page counterparts of the port 4 addresses are reserved for startup values.

---

<b>Important</b>	Some screen holes are different for the Apple IIe mouse. Refer to Appendix F.
------------------	---

---

**Table 9-5**  
Mouse port screen hole locations

---

**Scratch area**

---

Location	Description
\$0478	Low byte of clamping minimum
\$04F8	Low byte of clamping maximum
\$0578	High byte of clamping minimum
\$05F8	High byte of clamping maximum

---

**Port 4 screen holes**

---

Location	Description
\$047C	Low byte of X coordinate
\$04FC	Low byte of Y coordinate
\$057C	High byte of X coordinate
\$05FC	High byte of Y coordinate
\$067C	Reserved
\$06FC	Reserved
\$077C	Status byte
	<b>Bit      1 Equals</b>
	7      Button down
	6      Button was down on last read and still down
	5      Movement since last read
	4      Reserved
	3      Interrupt from VBLInt
	2      Interrupt from button
	1      Interrupt from movement
	0      Reserved
\$07FC	Mode byte (current mode; mask out bits 4–7 when testing)
	<b>Bit      1 Equals</b>
	7-4    Reserved
	3      VBLInt active
	2      VBL interrupt on button
	1      VBL interrupt on movement
	0      Mouse active

---

**Port 5 screen holes**

---

Reserved

---

**Memory expansion**

The screen hole addresses for the mouse in the Apple IIc that supports the memory expansion card all end with F, instead of C; this is because the mouse has moved to slot 7 from slot 4. For example, the low byte of the X coordinate is stored at \$047C in the original and UniDisk 3.5 IIc's, while it is stored at \$047F in the memory expansion IIc.

---

---

**Using the mouse as a hand controller**

You can use the mouse as if it were a set of hand controllers or an X-Y pointing device in port 4. If you turn the mouse on, the Monitor hand controller (game paddle) routines take input from the mouse. This is possible because the mouse and the hand controllers all use the same back panel connector.

You can run a BASIC program that uses the Pdl function to read from the mouse by doing the following, either from the keyboard or from a program:

1. Start up the system with the BASIC program that uses paddles.
2. Type PR#4 and press Return to turn on the mouse.
3. Press Control-A Return to initialize the mouse.
4. Type PR#0 and press Return to restore output to the screen.
5. Run the program.

Play the game using the mouse instead of the paddles.

---

**Important**

Many copy-protected games do not work with a mouse. In addition, many games don't use built-in firmware for the paddles.

---

---

---

**Game input**

The Apple IIc supports game paddles, joysticks, and other hand controllers connected to the DB-9 connector on its back panel. Table 9-6 is a summary of game input characteristics.

**Table 9-6**  
Game Input characteristics

---

<b>Port number</b>	None.
<b>Commands</b>	None.
<b>Initial characteristics</b>	Game inputs cannot be disabled.

**Hardware page locations**

\$C061	Switch input 0 and Open Apple.
\$C062	Switch input 1 and Solid Apple.
\$C063	Mouse button (sense is opposite that of \$C061 to distinguish it from paddle button).
\$C064	Analog input (paddle) 0.
\$C065	Analog input (paddle) 1.
\$C070	Trigger paddle timer.

**Monitor firmware routines**

Location	Name	Description
\$FB1E	PRead	Reads a paddle position.

**I/O firmware entry points**

None.

**Use of screen holes**

None.

---

## The hand controller connector signals

Several inputs are available to programs or devices from the 9-pin D-type miniature connector on the back of the Apple IIc: two 1-bit inputs, or *switches*, and two analog inputs.

When you connect a pair of hand controllers to the 9-pin connector, the rotary controllers use two analog inputs, and the pushbuttons use two 1-bit inputs. However, you can also use these inputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick.

Complete electrical specifications of these inputs are given in Chapter 11; Table 11-22 shows the connector pin numbers.

## Switch Inputs (Sw0 and Sw1)

The two 1-bit inputs can be connected to the output of another electronic device that meets the electrical requirements described in Chapter 11, or to a pushbutton. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. From machine language, you can do a Branch Plus or Branch Minus on the state of bit 7. From BASIC, you can read the switch with a PEEK and compare the value with 128. If the value is 128 or greater, the switch is on.

The memory locations for these switches are \$C061, \$C062, and \$C063 (decimal locations 49249 through 49251), as shown in Table 9-6. Switch 0 and switch 1 are permanently connected to Open Apple and Solid Apple on the keyboard; these are the ones connected to the buttons on the hand controllers. Location \$C063 is a second address for the mouse button, so that a program can distinguish it from an Open Apple keypress. When the mouse button is pressed, \$C063 (bit 7) goes from 1 to 0, and \$C061 (bit 7) goes from 0 to 1.

## Analog Inputs (Pdl0 and Pdl1)

The two analog inputs are designed for use with 150-K $\Omega$  variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit. The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

A program must first reset the timing circuits before it can read the analog inputs. Accessing memory location \$C070 does this. As soon as you reset the timing circuits, the high bits of the bytes at locations \$C064 through \$C067 are set to 1. If you PEEK at them from BASIC (locations 49252 through 49255), the values will be 128 or greater. Within about three milliseconds, these bits will change back to 0—byte values less than 128—and remain there until you reset the timing circuits again. The exact amount of time each of the bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

---

## Monitor support for game input

To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to 0, or you can use the built-in routine PRead. BASIC and other high-level languages also include convenient means of reading the analog inputs—refer to your language manuals. You can read and reread the same paddle at arbitrarily short intervals. However, you must wait at least three milliseconds between reading one paddle and reading a different paddle.

The Monitor routine PRead (at address \$FB1E) places in the Y register a number between \$00 and \$FF that represents the position of a hand controller. You pass the number of the hand controller in the X register.

---

**Warning** If the hand controller number you furnish in the X register does not equal 0 or 1, strange things may happen.

The paddle and vertical blanking both use \$C070. Disable interrupts before calling PRead if you are reading the paddles and using VBL interrupts.

---





## **Chapter 10**



### **Using the Monitor**

The System Monitor is a set of subroutines in the Apple IIc firmware that provides a standard interface to the built-in I/O devices described in Chapter 1. Many of the I/O subroutines described in Chapters 3 through 9 are part of the System Monitor.

DOS (but not ProDOS) and the BASIC interpreters (Appendix E) use these subroutines by direct calls to their starting locations. You can call the standard subroutines from your programs in the same fashion. The starting addresses for all of the standard subroutines are listed in Appendix C.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor

- ☐ to look at one or more memory locations
- ☐ to change the contents of any location
- ☐ to write small programs in machine language to be executed directly by the Apple IIc
- ☐ to move and compare blocks of memory
- ☐ to invoke other programs from the Monitor

---

---

## Invoking the Monitor

The System Monitor starts at memory location \$FF69 (–151). To invoke the Monitor, you make a CALL –151 statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompting character, an asterisk (\*), appears on the left side of the display screen, followed by a cursor.

To use the Monitor, you type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing Control-Reset, by pressing Control-C and then Return, or by typing 3D0G, which executes the resident program—usually Applesoft—whose address is stored in a jump instruction at location \$03D0.

The positive and negative decimal equivalents of Monitor locations are listed in Appendix C. In addition, Appendix H contains conversion tables from one numbering system to another. Appendix E gives further details on how to use Apple IIc firmware from BASIC programs.

### Important

---

If ProDOS (or DOS) is connected via the standard I/O links (Chapter 3), then you can issue commands to it from the Monitor. Under this arrangement, errors will return control to BASIC rather than to the Monitor.

---

If you want to have Control-Reset return you to the Monitor, load the values \$69, \$FF, and \$5A (decimal 105, 255, and 90) into the three locations starting at address \$03F2 (decimal 1010, the reset-vector address and the power-up byte).

---

---

## Syntax of Monitor commands

To give a command to the Monitor, you type a line on the keyboard, then press Return. The Monitor accepts the line using the standard I/O subroutine GetLn described in Chapter 3. A Monitor command can be up to 255 characters in length, ending with a carriage return. It can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading 0's; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

Each command you type consists of one command character, usually the first letter of the command name. The Monitor recognizes 22 different command characters. Some of them are punctuation marks, some are letters (uppercase or lowercase), and some are control characters.

❖ *Note:* Although the Monitor recognizes and interprets them, control characters typed on an input line do not appear on the screen.

This chapter contains examples of Monitor command use. Some of the data values displayed by your Apple IIc may differ from the values printed in these examples, because they are variables stored in RAM.

---

---

## Monitor memory commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the *last opened location* and the *next changeable location*.

---

**Warning** Because locations \$C000 through \$C0FF contain special hardware circuits, issuing any command that reads or writes on this page can have unpredictable, and perhaps disastrous, results.

---

---

## Examining memory contents

When you type the address of a memory location and press Return, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

```
*E000
E000- 4C
*33
0033- AA
*
```

Each time the Monitor displays the value stored at a location, it saves that address as the last opened location and as the next changeable location.

---

## Memory dump

When you type a period (.) followed by an address, and then press Return, the Monitor displays a memory dump: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger the address after the period is than the last opened location.

```
*20
0020- 00
*.2B
0021- 28 00 18 0F 0C 00 00
0028- A8 06 D0 07
*300
0300- 99
*.315
0301- B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03
*.32A
0316- 85 41
0318- 84 40 8A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20
*
```

When the Monitor performs a memory dump, it starts at the address immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a multiple of 8—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a *memory range*.

```
*300.32F
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
*30.40
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
*E015.E025
E015- 4C ED FD
E018- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing Return by itself makes the Monitor display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-eight boundary. The Monitor saves the address of the last location displayed as both the last opened location and the next changeable location.

```
*5
0005- 00
*Return
00 00
*Return
0008- 00 00 00 00 00 00 00 00
*32
0032- FF
*Return
AA 00 C2 05 C2
*Return
0038- 1B FD D0 03 3C 00 3F 00
*
```

---

## Changing memory contents

The section on memory dumping showed you how to *display* values stored in the Apple IIc's memory; this section shows you how to *change* these values. You can change any location in RAM; you can change the characteristics and treatment of an output device by changing the contents of locations assigned to it; and you can change a soft switch setting by referencing its set and reset addresses.

---

**Warning** Use these commands carefully. If you change the zero-page locations used by the interpreter or operating system (Appendix B), you may lose programs or data stored in memory.

---

### Changing one byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, you open location 0, then type a colon followed by a value.

```
*0
0000- 4C
*:5F
```

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

```
*0
0000- 5F
*
```

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the next example, you type the address again to verify the change.

```
*302:42
*302
0302- 42
*
```

When you change the contents of a location, the value that was contained in that location is replaced by the new value, which will remain until you or some program replaces it with another value.

❖ *ASCII input mode:* The Monitor has a tool to make entering values a little easier: ASCII input mode. ASCII input mode lets you enter ASCII characters as well as their hexadecimal ASCII equivalents. This means that 'A' is the same as C1 and 'B' is the same as C2 to the Monitor. The ASCII value for *any* character following an apostrophe is used by the Monitor. For example, to enter the string "Good morning!" at \$0300 in memory, type

```
*300:'G 'o 'o 'd ' 'm 'o 'r 'n 'i 'n 'g '!
```

Note that each character to be placed in memory is delimited by a leading and a trailing space. The only exception to this rule is that the last character in the line is followed by a return character instead of a space.

## Changing consecutive locations

You don't have to type a separate command with an address, a colon, a value, and press Return for each location you want to change. You can change the values of up to 85 consecutive locations at a time—or even more, if you omit leading 0's from the values—by typing only the initial address and colon followed by all the values separated by spaces; end with Return. The Monitor stores the consecutive values in consecutive locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next changeable location. Thus, you can continue changing consecutive locations, without typing an address on the next input line, by typing another colon and more values.

In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 03
*300
0300- 69
*Return
01 20 ED FD 4C 00 03
*10:0 1 2 3
*:4 5 6 7
*10.17
0010- 00 01 02 03 04 05 06 07
*
```

---

## Moving data in memory

You can copy a contiguous block of data from one area in the Apple IIc's memory to another in RAM by using the Monitor's MOVE command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory—the source locations—and where you want the copy to go—the destination locations.

The format of the complete MOVE command looks like this:

***{destination} < {start} . {end} M***

The destination is the address where you want the first of the moved data to go. The less-than symbol (<) separates the destination address from the starting and ending addresses of the block of data to be moved. The period between two addresses is the Monitor's standard notation for specifying address ranges. If the second address in the source range specification is less than the first, then only one value (that of the first location in the range) will be moved.

When you type the actual command, replace the words in braces with hexadecimal addresses, and omit the braces and spaces.

Here are some examples of memory moves. First, you examine the values stored in one range of memory, then store several values in another range of memory. The actual MOVE commands end with M.

```
*0.F
0000- 5F 00 05 07 00 00 00 00
0008- 00 00 00 00 00 00 00 00
*300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00 03
*300.30C
0300- A9 8D 20 ED FD A9 45 20
0308- DA FD 4C 00 03
*0<300.30C M
*0.C
0000- A9 8D 20 ED FD A9 45 20
0008- DA FD 4C 00 03
*310<8.A M
*310.312
0310- DA FD 4C
*2<7.9 M
*0.C
0000- A9 8D 20 DA FD A9 45 20
0008- DA FD 4C 00 03
*
```

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the last location in the source range as the last opened location, and the first location in the source range as the next changeable location.

If the destination address of the MOVE command is inside the source range of addresses, then strange things happen: the locations between the beginning of the source range and the destination address are treated as a subrange and the values in this subrange are replicated throughout the source range. Try it.

---

## Comparing data in memory

You can use the VERIFY command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the VERIFY command can be used immediately after a MOVE to make sure that the move was successful.

The VERIFY command, like the MOVE command, needs a range and a destination. The syntax of the VERIFY command is

```
{destination} < {start} . {end} V
```

See "Advanced Operations" for an interesting application of this feature.

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to \$0D, copy them to locations starting at \$0300 with the MOVE command, and then compare them using the VERIFY command. When you use the VERIFY command after you change the value at location 6 to \$E4, it detects the change.

```
*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4 C5
*300<0.D M
*300<0.D V
*6:E4
*300<0.D V
0006-E4 (EE)
*
```

If the VERIFY command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, VERIFY displays nothing. The VERIFY command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the MOVE command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges are compared. Like the MOVE command, the VERIFY command does unusual things if the destination address is within the source range; see “Advanced Operations” later in this chapter.

---

---

## Monitor register commands

Even though the actual contents of the 65C02's internal registers are changing as you use the Monitor, you can examine the values that the registers contained at the time the Monitor gained control, either because you called it or because the program you are debugging stopped at a break (BRK). You can also store new register values that will be used when you execute a program from the Monitor using the GO command, described below.

---

## Changing registers

When you call the Monitor, it stores the contents of the 65C02 registers in memory. The registers are stored in the order A, X, Y, P (processor status register), and S (stack pointer), starting at location \$45. When you give the Monitor a GO command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

---

## Examining registers

Pressing Control-E and then Return invokes the Monitor's EXAMINE command, which displays the stored register values and sets the location containing the contents of the A register as the next changeable location. After using the EXAMINE command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

```
*Control-E
M=00 A=0A X=FF Y=D8 P=B0 S=F8
*:B0 02
*Control-E
M=00 A=B0 X=02 Y=D8 P=B0 S=F8
*
```

In the EXAMINE command's display, M shows the current memory state register contents. The memory state register is location \$44, and its interpretation is given in Appendix E.

---

---

## Miscellaneous Monitor commands

Monitor commands discussed in this section let you do the following:

- ☐ change the video display format from normal to inverse and back
- ☐ assign input and output to various devices
- ☐ leave the Monitor and return to the currently loaded operating system (DOS 3.3 or ProDOS) or BASIC

The COut subroutine is described in Chapter 3.

---

## Display inverse and normal

You can control the setting of the inverse-normal mask location used by the COut subroutine from the Monitor so that all the Monitor's output will be in inverse format. The INVERSE command (I) sets the mask so that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the NORMAL command (N).

```
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*I
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*N
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*
```

See Appendix D.

---

## Back to BASIC

If you are using one of the Apple disk operating systems (ProDOS or DOS), press Control-Reset or type 3D0G to return to the language you were using, with your program and variables intact.

### Important

If you type 3D0G, make sure that the third character you type is a zero, not a letter O. The letter G is the Monitor's GO command.

If there is no operating system in RAM, use the BASIC command Control-B to leave the Monitor and enter the BASIC interpreter that was active when you entered the Monitor. (Normally this is Applesoft BASIC.) Any program or variables that you previously had in BASIC will be lost. If you want to reenter BASIC with your previous program and variables intact, use the CONTINUE BASIC command (Control-C).

---

## Redirecting input and output

The Control-P command diverts all output normally destined for the screen (port 0) to a device attached to one of the other ports, from 1 to 7. The format of the command is

*{port number}* Control-P

A Control-P command to port number 0 switches the stream of output characters back to the Apple IIc's video display. However, use Escape Control-Q if the enhanced video firmware is active (solid-block cursor).

Control-K controls the input stream in much the same way that Control-P controls the output stream. The format for the command is

*{port number}* Control-K

Pressing O Control-K directs the Monitor to accept input from the Apple IIc's built-in keyboard.

The Control-P and Control-K commands are the exact equivalents of the BASIC (but not DOS and ProDOS) commands PR# and IN#.

---

## Hexadecimal arithmetic

You can use the Monitor as a one-byte hexadecimal addition and subtraction calculator. Just type a line in one of these formats followed by Return:

*{value}* + *{value}* Return *{value}* - *{value}* Return

The Apple IIc performs the arithmetic and displays the result, as shown in these examples.

```
*20+13
=33
*4A-C
=3E
*
```

Chapter 3 lists the Apple IIc port numbers available.

For more information on the way those commands work, refer to "The Standard I/O Links" in Chapter 3.

---

---

## Advanced operations

This section describes some ways of using the Monitor commands to speed up your work.

---

### Multiple-command lines

You can put as many Monitor commands on a single line as you like, as long as you separate them with spaces, and the total number of characters in the line is less than 254. Adjacent single-letter commands such as L, S, I, and N need not be separated by spaces.

You can freely intermix all of the commands except the STORE (:) command. Because the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a STORE must be followed by a letter command before another address is encountered. You can use the NORMAL command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300,307 300:18 69 1 N 300.302
0300- 00 00 00 00 00 00 00 00
0300- 18 69 01
*
```

If the Monitor encounters a character in the input line that it does not recognize as either a hexadecimal digit or a valid command character, it executes all the commands on the input line up to that character, then stops with a beep and ignores the remainder of the input line.

---

### Filling memory

The MOVE command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range

```
*300:11 22 33
*
```

Remember the number of values in the pattern: in this case, it is three. Use the number to compute addresses for the MOVE command, like this:

*{start+number} < {start} . {end-number} M*

This MOVE command first replicates the pattern at the locations immediately following the original pattern, then replicates that pattern following itself, and so on until it fills the entire range.

```
*303<300.32D M
*300.32F
0300- 11 22 33 11 22 33 11 22
0308- 33 11 22 33 11 22 33 11
0310- 22 33 11 22 33 11 22 33
0318- 11 22 33 11 22 33 11 22
0320- 33 11 22 33 11 22 33 11
0328- 22 33 11 22 33 11 22 33
*
```

You can use the VERIFY command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example, to see the VERIFY command detect the discrepancy, you first fill the memory range from \$0300 to \$0320 with 0's and verify it, then change one location and verify again:

```
*300:0
*301<300.31F M
*301<300.31F V
*304:02
*301<300.31F V
0303-00 (02)
0304-02 (00)
*
```

---

## Repeating commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as N, and ending it with the sequence 34:n, where n is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, n=0. The value for n must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location \$0200. Each time the Monitor executes a command, it stores the value of the index at location \$34; when that command is finished, the Monitor reloads the index register with the value at location \$34. By making the last command change the value at location \$34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press Control-Reset; that is how this example ends.

```
*N 300 302 34:0 N
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
030
*
```

---

## Creating your own commands

The USER command (Control-Y) forces the Monitor to jump to memory location \$03F8. You can put a JMP instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a program that displays everything on the input line after the Control-Y. The program starts at location \$0300; the command line that starts with \$03F8 stores a jump to \$0300 at location \$03F8.

```
*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0 F5 4C 69 FF
*3F8:4C 00 03
*Control-Y THIS IS A TEST
THIS IS A TEST
*
```

---

---

## Machine-language programs

The main reason to program in machine language is to get more speed and sometimes to also save memory space. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

- ❖ *Note:* If you have never used machine language before, you'll need to learn the 65C02 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it, and study one of the books on 65C02 programming listed in the bibliography.

You can get a hexadecimal dump of your program or move it around in memory using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

---

### Running a program

The Monitor command to start execution of your machine-language program is the GO command. When you type an address and press G, the Apple IIc starts executing machine-language instructions starting at the specified location. If you just press G, execution starts at the last opened location. The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters A through Z: you store it starting at location \$0300, examine it to be sure you typed it correctly, then type 3D0G to start it running.

```
*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60
*300.30C
0300- A9 C1 20 ED FD 18 69 01
0308- C9 DB D0 F6 60
*300 G
ABCDEF GHIJKLMNOPQRSTUVWXYZ
*
```

---

## Disassembled programs

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called **assemblers**.

Programs like the Monitor's LIST command are called **disassemblers**. This command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the LIST command displays each instruction on a separate line, with a three-letter instruction name, or mnemonic, and a formatted hexadecimal operand. The LIST command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor LIST command has the format

*{location}* L

The LIST command starts at the specified location and displays as much memory as it takes to make up a screenful (20 lines) of instructions, as shown in the following example:

```
*300 L
0300-      A9 C1          LDA      #$C1
0302-      20 ED FD      JSR      $FDED
0305-      18            CLC
0306-      69 01          ADC      #$01
0308-      C9 DB          CMP      #$DB
030A-      D0 F6          BNE      $0302
030C-      60            RTS
030D-      00            BRK
030E-      00            BRK
030F-      00            BRK
0310-      00            BRK
0311-      00            BRK
0312-      00            BRK
0313-      00            BRK
0314-      00            BRK
0315-      00            BRK
0316-      00            BRK
0317-      00            BRK
0318-      00            BRK
0319-      00            BRK
*
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has 0's in it; other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the LIST command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the program counter, which it uses only to point to locations within programs. Whenever the Monitor performs a LIST command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another LIST command it displays another screenful of instructions, starting where the previous display left off.

---

---

## The STEP and TRACE commands

---

**Important** This section applies only to the UniDisk 3.5 and memory expansion versions of the Apple IIc.

---

STEP and TRACE are Monitor facilities for debugging assembly-language programs. The STEP command decodes, displays, and executes one instruction at a time, and the TRACE command steps continuously through a program, stopping when a BRK instruction is executed or Solid Apple is pressed. You can press Open Apple to slow down the trace to one step per second.

Each STEP command causes the Monitor to execute the instruction in memory pointed to by the program counter. The instruction is displayed in its disassembled form, then executed. The contents of the 65C02's internal registers are displayed after the instruction is executed. After execution, the program counter is incremented to point to the next instruction in the program.

Here is an example of the STEP command, using the following program:

```
$0300: LDX #02
$0302: LDA $00,X
$0304: STA $10,X
$0306: DEX
$0307: STA $C030
$030A: BPL $0302
$030C: BRK
```

To step through this program, first call the Monitor by typing `CALL -151` and pressing Return, and then from the Monitor type `300S` (to start the STEP routine at address `$0300`). Type `S` to advance each additional step through the program. The Monitor keeps the program counter and the last opened address separate from one another, so you can examine or change the contents of memory while you are stepping through your program. Here's what happens when you step through the program above, examining the contents of location `$0012` after the third step. Note that in this example, what you type appears just after the `*` prompt, and the information on the next two lines—that begin without the `*` prompt—is what the computer displays on the screen in response.

```
*300S
0300-  A2 02      LDX #02
M=CA A=0A X=02 Y=D8 P=30 S=F8
*S
0302-  B5 00      LDA $00,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*S
0304-  95 10      STA $10,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*12
0012- 0C
*S
0306-  CA          DEX
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
0307-  8D 30 C0    STA $C030
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
030A-  10 F6      BPL $0302
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
0302-  B5 00      LDA $00,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*S
0304-  95 10      STA $10,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*
```

The TRACE command is a continuous version of the STEP command; it stops stepping through the program only when you press Solid Apple, or when it encounters a BRK instruction in the program. Press Open Apple to slow the trace to one step per second.

- 
- Important** Keep the following cautions in mind when using the STEP and TRACE Monitor commands:
- ☐ If the program ends with an RTS instruction, the TRACE routine will continue to run indefinitely until stopped with Solid Apple.
  - ☐ You can't step or trace through routines that use the same zero page locations as the Monitor.
- 

---

---

## The Mini-Assembler

---

- Important** This section applies only to the UniDisk 3.5 and memory expansion versions of the Apple IIc.
- 

Without an assembler, you have to write your machine-language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the Monitor commands described earlier in this chapter.

The Mini-Assembler lets you enter machine-language programs directly from the keyboard of your Apple. ASCII characters can be entered in Mini-Assembler programs, exactly as you enter them in the Monitor.

Note that the Mini-Assembler doesn't accept labels; you must use actual values and addresses.

---

### Starting the Mini-Assembler

---

To start the Mini-Assembler, first invoke the Monitor by typing `CALL -151` and pressing Return, and then from the Monitor, type `!` followed by Return. The Monitor prompt character then changes from `*` to `!`.

When you finish using the Mini-Assembler, press Return from a blank line to return to the Monitor.

To enter code into memory, type the address, a colon, and the instruction. For example, after entering the Mini-Assembler, you could type

```
!300:STA C030
```

You can enter a series of instructions by typing a space, followed by the instruction, followed by Return:

```
! 300:STA C030
! LDA #A0
! INX
```

Each succeeding instruction is placed in the next available memory location. As you type in instructions, each is replaced by the starting address of the instruction, the hexadecimal value(s) of the instruction, followed by mnemonics describing the instruction. For example, the sequence of instructions given above would produce the following on your screen:

```
0300-    8D 30 C0        STA $C030
0303-    A9 A0          LDA #$A0
0305-    E8             INX
```

When you're ready to execute your program, press Return to leave the Mini-Assembler and return to the Monitor. Monitor commands can't be executed directly from the Mini-Assembler.

---

## Using the Mini-Assembler

The Mini-Assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the Mini-Assembler to store your program. Do this by typing the address followed by a colon.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction. Now press Return. The Mini-Assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the program counter, and then disassembles it again and displays the disassembled line. It then displays a prompt on the next line.

Now the Mini-Assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press Return. The Mini-Assembler assembles that line and waits for another.

If the line you type has an error in it, the Mini-Assembler beeps loudly and displays a caret (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, and so forth. The Mini-Assembler also rejects the input line if you forget the space before or after a mnemonic or include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the Mini-Assembler flags this as an error.

❖ *Dollar signs:* In this manual, dollar signs (\$) in addresses signify that the addresses are in hexadecimal notation. The dollar signs are ignored by the Mini-Assembler and can be omitted in programs.

```
!300:LDX #02
0300-   A2 02          LDX    #$02
! LDA $00,X
0302-   B5 00          LDA    $00,X
! STA $10,X
0304    95 10          STA    $10,X
! DEX
0306-   CA            DEX
! STA $C030
0307-   8D 30 C0       STA    $C030
! BPL $0302
030A-   10 F6          BPL    $0302
! BRK
030C-   00            BRK
!
```

To leave the Mini-Assembler and reenter the Monitor, press Return at a blank line.

Your assembly-language program is now stored in memory. You can display it with the LIST command:

```
*300L
0300-   A2 02          LDX    #$02
0302-   B5 00          LDA    $00,X
0304-   95 10          STA    $10,X
0306-   CA            DEX
0307-   8D 30 C0       STA    $C030
030A-   10 F6          BPL    $0302
030C-   00            BRK
030D-   00            BRK
030E-   00            BRK
030F-   00            BRK
0310-   00            BRK
```

0311-	00	BRK
0312-	00	BRK
0313-	00	BRK
0314-	00	BRK
0316-	00	BRK
0316-	00	BRK
0317-	00	BRK
0318-	00	BRK
0319-	00	BRK
*		

**Table 10-1**  
Mini-Assembler address formats

Addressing mode	Format
Accumulator	*
Implied	*
Immediate	#{ <i>value</i> }
Absolute	\${ <i>address</i> }
Zero page	\${ <i>address</i> }
Indexed zero page	\${ <i>address</i> },X \${ <i>address</i> },Y
Indexed absolute	\${ <i>address</i> },X \${ <i>address</i> },Y
Relative	\${ <i>address</i> }
Indexed indirect	(\${ <i>address</i> }),X
Indirect indexed	(\${ <i>address</i> }),Y
Absolute indirect	(\${ <i>address</i> })

\* These instructions have no operands.

## Mini-Assembler instruction formats

The Apple IIc Mini-Assembler recognizes 66 mnemonics and 15 addressing formats. The mnemonics are standard, as used in the *Synertek Programming Manual* (Apple part number A2L0003), but the addressing formats are somewhat different, as shown in Table 10-1.

An address consists of one or more hexadecimal digits. The Mini-Assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the Mini-Assembler adds leading 0's; if the address has more than four digits, then it uses only the last four.

There is no syntactical distinction between the absolute and zero-page addressing modes. If you give an instruction to the Mini-Assembler that can be used in both absolute and zero-page mode, the Mini-Assembler assembles that instruction in absolute mode if the operand for that instruction is greater than \$FF, and it assembles it in zero-page mode if the operand is less than \$0100.

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The Mini-Assembler calculates the relative distance to use in the instruction automatically. If the target address is more than 127 locations distant from the instruction, the Mini-Assembler sounds a bell (beep), displays a caret (^) under the target address, and does not assemble the line.

If you give the Mini-Assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, then the Mini-Assembler will not accept the line.

---

---

## Summary of Monitor commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one.

---

### Examining memory

<code>{adrs}Return</code>	Displays the value contained in one location.
<code>{adrs1}.{adrs2}Return</code>	Displays the values contained in all locations between <code>{adrs1}</code> and <code>{adrs2}</code>
<code>Return</code>	Displays the values in up to eight locations following the last opened location.
<code>{adrs}L</code>	Lists disassembled code starting at <code>{adrs}</code> and continuing until the screen is full.

---

### Changing the contents of memory

<code>{adrs}:{val}{val}...</code>	STORE command. Stores the values in consecutive memory locations starting at <code>{adrs}</code> .
<code>:{val}{val}...</code>	Stores values in memory starting at the next changeable location.

---

### Moving and comparing

<code>{dest}&lt;{start}.{end}M</code>	MOVE command. Copies the values in the range <code>{start}.{end}</code> into the range beginning at <code>{dest}</code> .
<code>{dest}&lt;{start}.{end}V</code>	VERIFY command. Compares the values in the range <code>{start}.{end}</code> to those in the range beginning at <code>{dest}</code> .

---

## The Register command

Control-E	EXAMINE command. Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.
-----------	---

---

## Miscellaneous Monitor commands

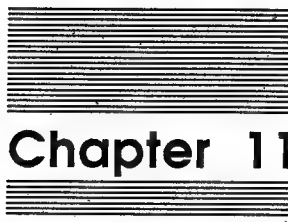
I	INVERSE command. Sets inverse display mode.
N	NORMAL command. Sets normal display mode.
Control-B	BASIC command. Enters the language currently active (normally Applesoft).
Control-C	CONTINUE BASIC command. Returns to the language currently active (normally Applesoft).
{val}+{val}	Adds the two values and prints the hexadecimal result.
{val}-{val}	Subtracts the second value from the first and prints the result.
{port}Control-P	Redirects output to the device connected to port number {port}. If {port}=0, sends output to the video display. Use only when the enhanced video firmware is not active (checkerboard cursor).
Escape Control-Q	Redirects output to video display when enhanced video firmware is active (solid block cursor).
{port}Control-K	Takes input from the device connected to port number {port}. If {port}=0, accepts input from the keyboard.
Control-Y	USER command. Jumps to the machine-language subroutine at location \$03F8.

---

## Running and listing programs

<code>{<i>addr</i>}G</code>	Transfers control to the machine-language program beginning at <code>{<i>addr</i>}</code> .
<code>{<i>addr</i>}L</code>	Disassembles and displays 20 instructions starting at <code>{<i>addr</i>}</code> . Subsequent L's display 20 more instructions each.





## **Chapter 11**

# **Hardware Implementation**

Most of this manual describes functions—what the Apple IIc does. This chapter, on the other hand, describes objects—the pieces of hardware the Apple IIc uses to carry out its functions. If you are designing a device to connect to the Apple IIc back panel, or if you just want to know more about how the Apple IIc is built, you should study this chapter.

---

---

## Environmental specifications

The Apple IIc is quite sturdy when used in the way it was intended: as a transportable computer, made for use in an indoor environment. You can carry it by its handle from room to room, but for longer trips you should use its carrying case or some other protective container (such as an attaché case). Table 11-1 defines the conditions under which the Apple IIc is designed to function properly.

**Table 11-1**  
Environmental specifications

<b>Operating temperature</b>	10° to 40° C (50° to 104° F)
<b>Relative humidity</b>	20% to 95%

You should treat the Apple IIc with the same kind of care as any other electrical appliance; protect it from physical abuse, and be careful not to bump it against furniture when you move it around. Put it in an attache case or other protective covering if you carry it outside. You should also protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids, particularly those with dissolved contaminants, such as soups, fruit juices, and carbonated soft drinks.

In normal operation (with the handle locked in its down position), enough air flows through the openings in the case to keep the insides from getting too hot. If you do overheat your Apple IIc—for example, by blocking the upper or lower ventilation openings—the first symptom will be erratic operation, such as unexpectedly changed data. (The memory devices in the Apple IIc are especially sensitive to heat.) Letting the machine cool down by turning it off for a while and unblocking the vents before using it again will bring it back to normal operation. The only exception to this is if you have gotten your Apple IIc *too* hot and physically damaged some internal component.

Disks are another heat-sensitive element of the system. If the built-in drive becomes too hot, a disk within can warp or even melt. A melted or warped disk can't be used again.

---

---

## Power requirements

The electrical power used by the Apple IIc—and everything that draws power from it—is limited by the capacities of the computer's power supply and internal voltage converter. This section describes these limits for the USA external power supply. Appendix G describes them for models built for other countries. The internal voltage converter is the same on all models.

---

### The external power supply

If you purchased your Apple IIc outside the USA, consult Appendix G for external power supply characteristics.

The external power supply operates on normal household AC power and provides DC power to the Apple IIc internal converter. The basic specifications of the external power supply are listed in Table 11-2. The Apple IIc external power supply's cord must be plugged into a three-wire 115-volt (nominal) outlet. A two-wire outlet is not properly grounded—using it will damage the external power supply and perhaps the Apple IIc as well. The line voltage must be in the range given in Table 11-2.

---

**Warning** Important safety instructions: This product is equipped with a three-wire grounding-type plug—a plug having a third (grounding) pin. This plug will only fit into a grounding-type AC outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

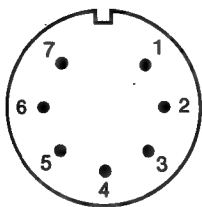
Do not defeat the purpose of the grounding-type plug.

---

**Table 11-2**  
Power supply specifications

---

<b>Line voltage</b>	105 to 129 VAC, 60 Hz
<b>Maximum power consumption</b>	25 W
<b>Supply voltage</b>	+15 VDC (nominal)
<b>Supply current</b>	1.2 A (nominal)



Pin	Signal
1	Not connected
2, 3	Signal ground
4	Shield ground
5, 6	+15 VDC
7	Not connected

**Figure 11-1**  
External power connector

## The external power connector

The external power supply is attached to the internal converter by means of a 7-pin DIN connector. The connector pins are identified in Figure 11-1 and Table 11-3.

**Table 11-3**  
External power connector signals

Pin	Signal	Description
1, 7		Not connected
2, 3	Ground	Common electrical ground
4	Chassis	Chassis ground
5, 6	+15V	+15-volt DC input to converter

## The internal converter

The internal converter in the Apple IIc operates with a supply voltage from 9 to 20 volts DC as provided by the external power supply or its equivalent. The internal converter provides enough low-voltage electrical power for the built-in electronics plus an external disk drive attached via the 19-pin connector. The basic specifications of the internal converter are listed in Table 11-4. Minus 5 volts is derived from the -12 volts (nominal) provided by the voltage converter.

**Table 11-4**  
Internal converter specifications

<b>Input voltage</b>	+9 to 20 VDC
<b>Maximum power consumption</b>	25 W
<b>Supply voltages</b>	+5V $\pm 5\%$ +12V $\pm 10\%$ -12V $\pm 10\%$
<b>Maximum supply currents</b>	+5V: 1.5 A +12V: 0.6 A continuous 0.9 A intermittent 1.5 A surge (for < 100 ms) -12V: 100 mA (-5V: 50 mA)
<b>Maximum case temperature</b>	60° C (140° F)

The Apple IIc uses a switching-type internal voltage converter as a power supply. It is small and lightweight, and it generates less heat than other types of voltage converters.

The voltage converter works by using the DC voltage input to power a variable-frequency oscillator. The oscillator drives a small transformer with several separate windings to produce the different voltages required. A circuit compares the voltage of the +5-volt supply with a reference voltage and feeds an error signal back to the oscillator circuit. The oscillator circuit uses the error signal to control the duty cycle of its oscillation and keep the output voltages in their normal ranges.

The converter includes circuitry to protect itself and the other electronic parts of the Apple IIc by limiting all three output voltages whenever it detects one of the following malfunctions:

- ☐ any supply voltage short-circuited to ground
- ☐ any output voltage outside the normal range

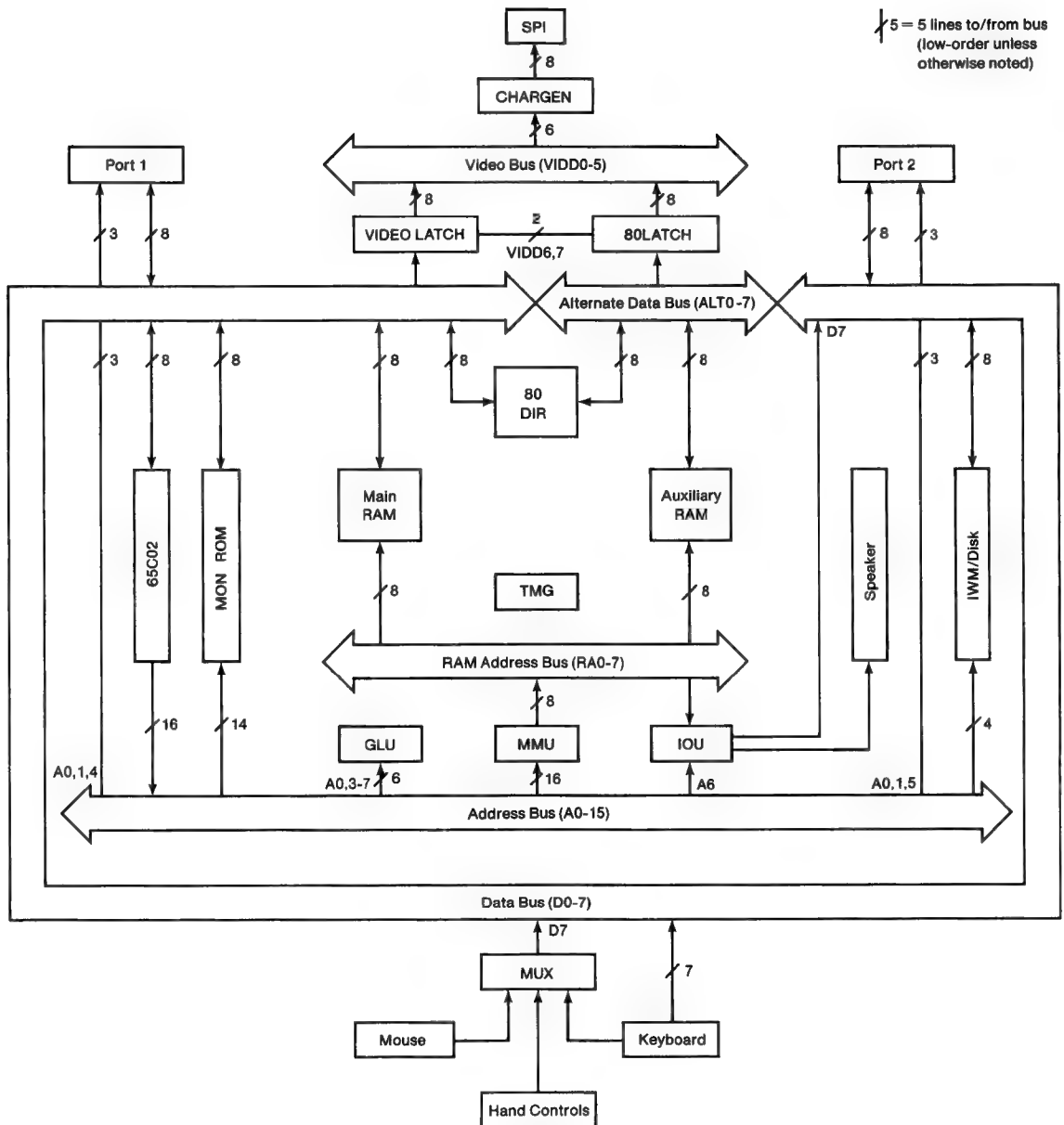
Whenever one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, and all the output voltages drop to 0 if they cannot be brought back into their normal range.

---

---

## **Apple IIc overall block diagram**

Figure 11-2 is an overall block diagram of the Apple IIc. The following sections contain more detailed diagrams of the major parts of the machine. A full set of schematic diagrams of the Apple IIc appears later in this chapter.



**Figure 11-2**  
Apple IIc block diagram

---

---

## The 65C02 microprocessor

**CMOS** (complementary metal-oxide semiconductor) is a way of making integrated circuits that require less power to operate than other technologies such as NMOS (negative-doped metal-oxide semiconductor), used by the 6502.

The Apple IIc uses a **CMOS** 6502 (designated as 65C02) microprocessor as its central processing unit (CPU). The 65C02 in the Apple IIc runs at a clock rate of 1.023 MHz and performs up to 500,000 8-bit operations per second.

❖ *Note:* The clock rate is not a very good criterion for comparing different types of microprocessors. The 65C02 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 65C02 with a 1-MHz clock is equivalent to many other types of microprocessors with clock rates up to 5 MHz.

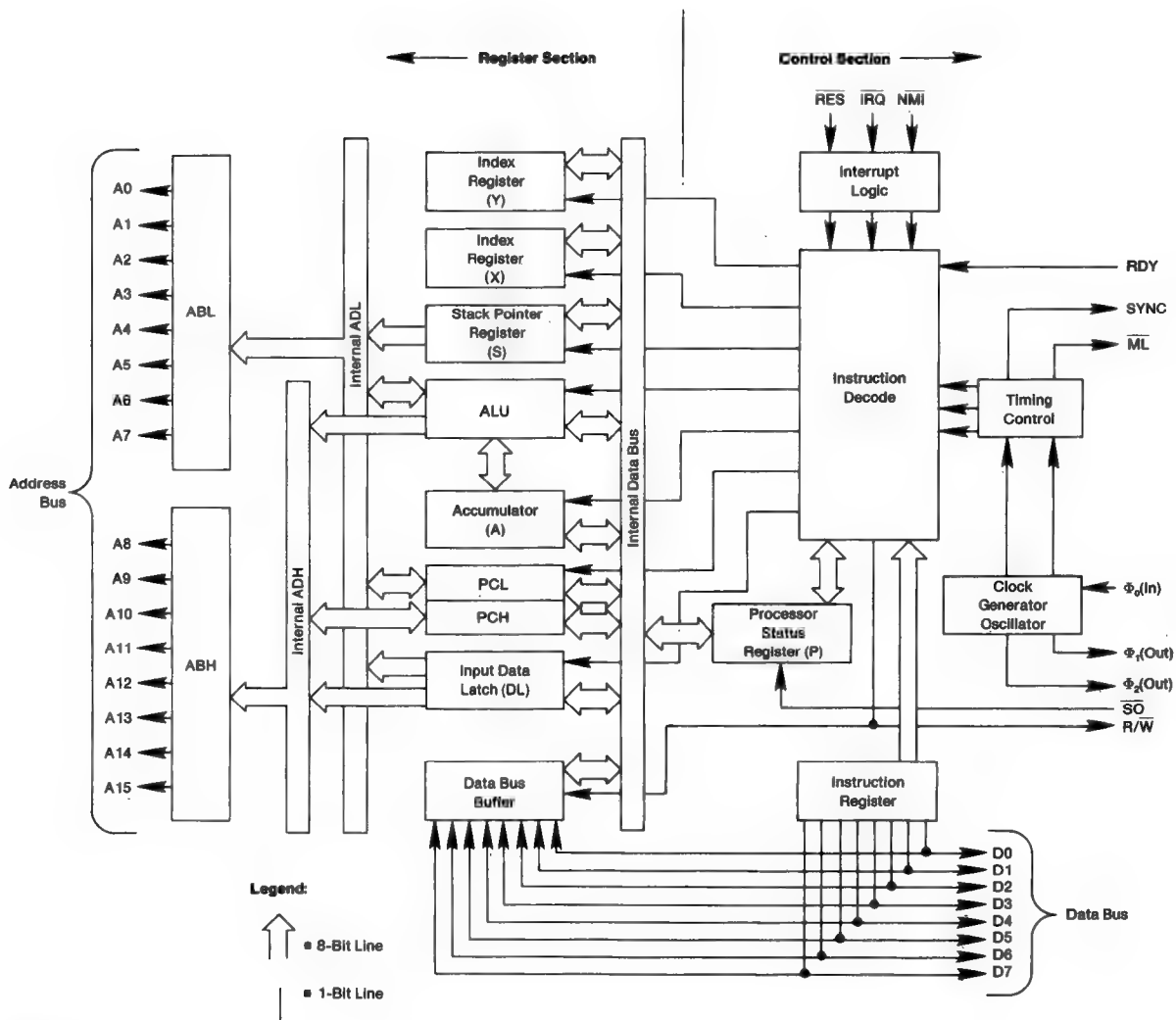
In addition to requiring less power than earlier NMOS 6502 processors, the 65C02 in the Apple IIc has 27 new instructions. However, programs that use these additional instructions are not backward compatible with other Apple II series computers that are not equipped with a CMOS 6502.

These instructions are described in Appendix A.

---

### 65C02 block diagram

Figure 11-3 is a block diagram of the 65C02 microprocessor. Table 11-5 contains the general specifications of this chip. The 65C02 has a 16-bit address bus, giving it an address space of 64K bytes. The Apple IIc uses special techniques to address a total of more than 64K (see Chapter 2).



**Figure 11-3**  
65C02 block diagram (copyright © 1982 by NCR Corporation; used by permission)

**Table 11-5**  
65C02 microprocessor specifications

---

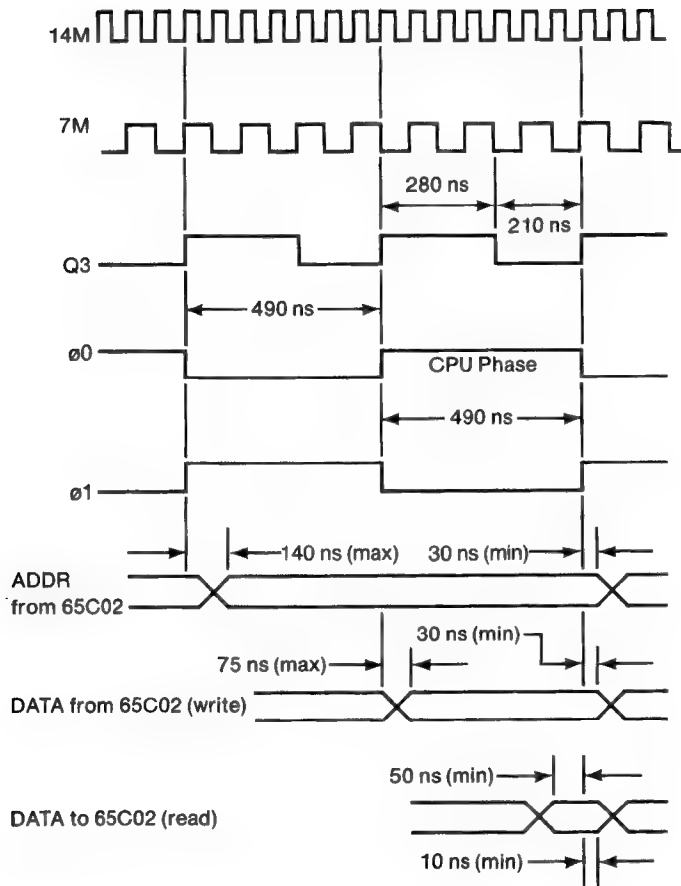
<b>Type</b>	65C02
<b>Register complement</b>	8-bit accumulator (A) 8-bit index registers (X,Y) 8-bit stack pointer (S) 8-bit processor status (P) 16-bit program counter (PC)
<b>Data bus</b>	8 bits wide
<b>Address bus</b>	16 bits wide
<b>Address range</b>	65,536 (64K)
<b>Interrupts</b>	IRQ (maskable) NMI (nonmaskable) BRK (programmed)
<b>Operating voltage</b>	+5V ( $\pm 5\%$ )
<b>Power dissipation</b>	5 mW (at 1 MHz)

---

## 65C02 timing

The Apple IIc's operation is controlled by a set of synchronous timing signals, sometimes called *clock signals*. The Apple IIc uses a 14.318-MHz master timing signal, called *14M*, to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. The timing signals directly involved with the 65C02's operation are described in this section. Other timing signals are described later in this chapter.

The relationships of the main 65C02 timing signals are diagrammed in Figure 11-4, and the signals are listed in Table 11-6. The 65C02 clock signals are  $\phi 1$  and  $\phi 0$ , complementary signals at a frequency of 1.0227 MHz. The Apple IIc signal  $\phi 0$  is similar to the signal  $\phi 2$  in Appendix A (it isn't identical—it's a tiny bit early).



**Figure 11-4**  
65C02 timing signals

**Table 11-6**  
65C02 timing signal descriptions

Signal	Description
14M	Master oscillator, 14.318 MHz; also 80-column dot clock
VID7M	Intermediate timing signal and 40-column dot clock
Q3	Intermediate timing signal, 2.045 MHz with asymmetrical duty cycle
ø0	Phase 0 of 65C02 clock, 1.0227 MHz; complement of ø1
ø1	Phase 1 of 65C02 clock, 1.0227 MHz; complement of ø0

The 65C02's operations are related to the clock signals in a simple way: internal during  $\phi 1$ , external during  $\phi 0$ . The 65C02 puts an address on the address bus during  $\phi 1$ . This address is valid not later than 110 nanoseconds after  $\phi 1$  goes high and remains valid through all of  $\phi 0$ . The 65C02 reads or writes data during  $\phi 0$ . If the 65C02 is writing, the read/write signal is low during  $\phi 0$  and the 65C02 puts data on the data bus. The data are valid not later than 75 nanoseconds after  $\phi 0$  goes high. If the 65C02 is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of  $\phi 0$ .

More information about the 65C02 and its instruction set is in Appendix A.

---

---

## The custom integrated circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIc is in five custom integrated circuits:

- ☐ the memory management unit (MMU)
- ☐ the input-output unit (IOU)
- ☐ the timing generator (TMG)
- ☐ the general logic unit (GLU)
- ☐ the disk controller unit, also known as the Integrated Woz Machine (IWM)

The soft switches that control the various I/O and addressing modes of the Apple IIc are addressable flags inside the MMU, IOU, and GLU. The functions of the MMU and IOU are not as independent as their names suggest; working together, they generate all the addressing signals. For example, the MMU generates the RAM address signals for the CPU, while the IOU generates similar RAM address signals for the video display and most I/O hardware addresses.

---

## The memory management unit (MMU)

The circuitry inside the MMU implements these soft switches:

- ☐ Page 2 display (Page2) (described in Chapter 5)
- ☐ high-resolution mode (HiRes) (Chapter 5)
- ☐ store to 80-column display (80Store) (Chapter 5)
- ☐ select bank 2 (Bank2) (Chapter 2)

- enable bank-switched RAM (EnlCRAM) (Chapter 2)
- read auxiliary memory (RAMRd) (Chapter 2)
- write auxiliary memory (RAMWrt) (Chapter 2)
- auxiliary stack and zero page (AltZP) (Chapter 2)
- reset mouse Y interrupt (RstYInt) (Chapter 9)
- reset mouse X interrupt (RstXInt) (Chapter 9)

These switches are available on MMU pin 21, which is connected to bit 7 on the data bus. Figure 11-5 shows the MMU pinouts; Table 11-7 describes the signals.

**Important** A signal name followed by an asterisk is active low—that is, it is true when the signal is at a TTL high (+5V) level.

The 64K dynamic RAMs used in the Apple IIc use a multiplexed address, as described later in this chapter. The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU.

**Table 11-7**  
MMU signal descriptions

Pin	Signal	Description
1	GND	Power and signal common
2	A0	65C02 address input
3	ø0	Clock phase 0 input
4	Q3	Timing signal input
5	PRAS*	Memory row-address strobe
6–13	RA0–RA7	Multiplexed address output
14	R/W*	65C02 read-write control signal
15	INH*	Inhibits main memory (tied to +5V)
16	C06X*	Causes \$C06x outputs to go to 0 during ø0
17	EN80*	Enables auxiliary RAM
18	KBD*	Enables keyboard data bits 0-6
19	ROMEN2*	Enables ROM (tied to ROMEN1*)
20	ROMEN1*	Enables ROM (tied to ROMEN2*)
21	MD7	State of MMU flags on data bus bit 7
22	C07X	Causes \$C07x outputs to go to 0 during ø0
23	CASEN*	Enables main RAM
24	SELIO*	Goes to 0 during ø0 for any access to \$C0 page except \$C08x, Bx, Cx, or Fx
25	+5V	Power
26–40	A15–A1	65C02 address input

GND	1	40	A1
A0	2	39	A2
ø0	3	38	A3
Q3	4	37	A4
PRAS*	5	36	A5
RA0	6	35	A6
RA1	7	34	A7
RA2	8	33	A8
RA3	9	32	A9
RA4	10	31	A10
RA5	11	30	A11
RA6	12	29	A12
RA7	13	28	A13
R/W*	14	27	A14
INH*	15	26	A15
C06X*	16	25	+5V
EN80*	17	24	SELIO*
KBD*	18	23	CASEN*
ROMEN2*	19	22	C07X*
ROMEN1*	20	21	MD7

**Figure 11-5**  
MMU pinouts

## The input/output unit (IOU)

Input/output unit (IOU) implements the following soft switches, all described in Chapters 2 and 3:

- ☐ Page 2 display (Page2)
- ☐ high-resolution mode (HiRes)
- ☐ text mode (TEXT)
- ☐ mixed mode (MIXED)
- ☐ 80-column display (80Col)
- ☐ character-set select (AltChar)
- ☐ any-key-down (AKD)
- ☐ mouse movement (X0, Y0)
- ☐ vertical blanking interrupt (VBIInt)

These switches are available on IOU pin 9, which is connected to bit 7 on the data bus. Figure 11-6 shows the IOU pinouts; Table 11-8 describes the signals.

The 64K dynamic RAMs used in the Apple IIc require a multiplexed address, as described later in this chapter. The IOU generates this multiplexed address during clock phase 1 for the data transfers required for display and memory refresh. The way this address is generated is described under "The Video Counters" in this chapter.

**Table 11-8**  
IOU signal descriptions

Pin	Signal	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3	SEGA	In text mode, works with VC (see pin 5) and SEGB to determine character row address
4	SEGB	In text mode, works with VC (see pin 5) and SEGA; in graphics mode, selects high resolution when low, low resolution when high
5	VC	Displays vertical counter bit: in text mode, SEGA, SEGB, and VC determine which of the eight rows of a character's dot pattern to display; in low resolution, selects upper or lower block defined by a byte

GND	1	40	H0
GR	2	39	SYNC*
SEGA	3	38	WNDW*
SEGB	4	37	CLRGAT*
VC	5	36	RA10*
80COL*	6	35	RA9*
CASSO	7	34	VIDD6
SPKR	8	33	VIDD7
MD7	9	32	KSTRB
YMOVE	10	31	AKD
(N.C.)	11	30	IOUSELio*
(N.C.)	12	29	A6
PDL0/XMOVE	13	28	+5V
R/W*	14	27	Q3
RESET*	15	26	ø0
IRQ*	16	25	PRAS*
RA0	17	24	RA7
RA1	18	23	RA6
RA2	19	22	RA5
RA3	20	21	RA4

**Figure 11-6**  
IOU pinouts

**Table 11-8** (continued)  
IOU signal descriptions

Pin	Signal	Description
6	80COL*	80-column video enable
7	CASSO	Reserved
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)
10	YMOVE	Detects mouse movement along Y axis
11	N.C.	Not used
12	N.C.	Not used
13	PDL0/XMOVE	Detects mouse movement along X axis
14	R/W*	65C02 read-write control signal
15	RESET*	Power on and reset output
16	IRQ*	Maskable interrupt line to 65C02
17-24	RA0–RA7	Video refresh multiplexed RAM address (phase 1)
25	PRAS*	Row-address strobe (phase 0)
26	ø0	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 65C02
30	IOUSELIO*	Derived from the SELIO* output for MMU pin 24
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33,34	VIDD7,VIDD6	Video display data bits
35,36	RA9*,RA10*	Video display control bits
37	CLRGAT*	Color-burst gate (enable)
38	WNDW*	Displays blanking signal
39	SYNC*	Displays synchronization signal
40	H0	Displays horizontal timing signal (low bit of character counter)

14M	1	20	+5V
7M	2	19	PRAS*
CREF	3	18	(N.C.)
H0	4	17	PCAS*
VIDD7	5	16	Q3
SEGB	6	15	ø0
TEXT	7	14	ø1
CASEN*	8	13	VID7M
80COL*	9	12	LDPS*
GND	10	11	TMGEN*

**Figure 11-7**  
TMG pinouts

## The timing generator (TMG)

A custom timing generator chip (TMG) generates several timing and control signals in the Apple IIc. The TMG pinouts are shown in Figure 11-7; the signals are listed in Table 11-9.

**Table 11-9**  
TMG signal descriptions

Pin	Signal	Description
1	14M	14.318-MHz master timing signal input
2	7M	7.159-MHz timing signal
3	CREF	3.5795-MHz color reference timing signal
4	H0	Horizontal video timing signal
5	VIDD7	Video data bit 7
6	SEGB	Video timing signal
7	TEXT	Video display text-modes enable
8	CASEN*	RAM enable (CAS enable)
9	80COL*	Enables 80-column display mode
10	GND	Power and signal common
11	TMGEN*	Enables master timing
12	LDPS*	Video shift-register load enable
13	VID7M	Video dot clock enable, 7 MHz or continuous 0
14	ø1	Phase 1 system clock
15	ø0	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS*	RAM column-address strobe
18	N.C.	Reserved for testing
19	PRAS*	RAM row-address strobe
20	+5V	Power

## The general logic unit (GLU)

The general logic unit is a single chip that contains the miscellaneous logic required for the system. It provides

- ☐ all RAM read/write timing
- ☐ double high-resolution enable/disable
- ☐ soft-switch status registers
- ☐ write command registers
- ☐ IOU control for mouse interrupts
- ☐ double high-resolution soft switches

14M	1	24	+5V
A0	2	23	SER*
A3	3	22	IOUHOLE
A4	4	21	DISK*
A5	5	20	7M
A6	6	19	CREF
A7	7	18	(N.C.)
ø0	8	17	(N.C.)
SELIO*	9	16	TEXT
GR	10	15	R/W*
RESET*	11	14	MD7
GND	12	13	GLUEN*

**Figure 11-8**  
GLU pinouts

The GLU's pin assignments are shown in Figure 11-8 and its signals are listed in Table 11-10.

**Table 11-10**  
GLU signal descriptions

Pin	Signal	Description
1	14M	Master clock (14.318 MHz)
2,3-7	A0,A3-A7	Address lines to select least significant byte of addresses on C0 page
8	ø0	Phase 0 of 1.0227-MHz processor sync clock
9	SELIO*	Device select for selecting most significant byte of the address
10	GR	Graphics mode select line
11	RESET*	Master reset for system; resets GLU
12	GND	Ground reference and negative supply
13	GLUEN*	Enables GLU
14	MD7	Indicates status of MMU flags on data bus bit 7
15	R/W*	Read/write qualifier input from processor
16	TEXT	Signal used to generate video timing in double high-resolution or not-graphics
17,18	N.C.	Not used
19	CREF	Color reference signal
20	7M	7-MHz clock output
21	DISK*	Disk controller device select output
22	IOUHOLE	Controls IOUSELIO
23	SER*	Serial controller device select output
24	+5V	+5 volt supply

## The disk controller unit (IWM)

The IWM (for Integrated Woz Machine) is a disk controller that includes, on a single chip, all the capabilities of the disk controller card originally designed by Steve Wozniak in 1977.

Right after reset, the IWM is an integrated **GCR** (group code recording) disk drive controller. It also has a status register, mode register, and multiple operating modes. It provides both synchronous and asynchronous modes, and a fast mode with a data rate twice that of normal disk I/O speeds. Figure 11-9 shows the IWM pin assignments; Table 11-11 describes the IWM signals.

**Table 11-11**  
IWM signal descriptions

Pin	Signal	Description
1	SEEKPH0	Stepper motor control phase 0, one of four programmable disk drive motor phase outputs.
2	SEEKPH2	Stepper motor control phase 2.
3	A0	The data input to the state bit selected by A1 to A3.
4–6	A1–A3	These three inputs select one of the eight bits in the state register to be updated.
7	DISK*	Device enable. The falling edge of DISK* latches information on A1 to A3. The rising edge of either Q3 or DISK* qualifies write register data.
8	WRDATA	The serial data output. Each 1-bit causes a transition on this output.
9	WRREQ*	This signal is a programmable buffered output line.
10–13	D0–D3	D0 to D7 make up the bidirectional data bus.
14	GND	Ground reference and negative supply.
15–18	D4–D7	The remaining bits of the bidirectional data bus.
19	DR2*	Drive 2 select.

For further information on **GCR**, refer to "Disk I/O."

SEEKPH0	1	28	SEEKPH1
SEEKPH2	2	27	SEEKPH3
A0	3	26	+5V
A1	4	25	Q3
A2	5	24	7M
A3	6	23	RESET*
DISK*	7	22	RDDATA
WRDATA	8	21	WRPROT
WRREQ*	9	20	DR1*
D0	10	19	DR2*
D1	11	18	D7
D2	12	17	D6
D3	13	16	D5
GND	14	15	D4

**Figure 11-9**  
IWM pinouts

**Table 11-11** (continued)  
IWM signal descriptions

Pin	Signal	Description
20	DR1*	Drive 1 select.
21	WRPROT	Write-protect input; this can be polled via bit 7 of the status register.
22	RDDATA	Serial data input line. The IWM synchronizes the falling transition of each pulse.
23	RESET*	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to 0.
24	7M	7-MHz clock input.
25	Q3	A 2.0-MHz clock input used to qualify the timing of the serial data being written or read.
26	+5V	The +5 volt supply.
27	SEEKPH3	Stepper motor control phase 3.
28	SEEKPH1	Stepper motor control phase 1.

---

---

## Memory addressing

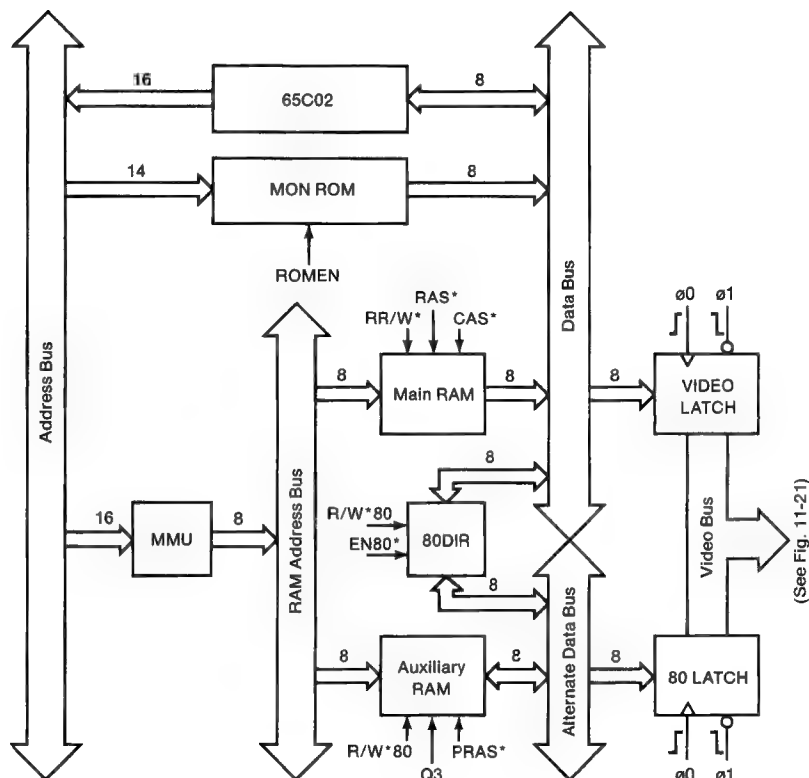
The 65C02 microprocessor can directly address 65,536 locations. The Apple IIc uses this entire address space, and then some: some areas in memory are used for more than one function. The following sections describe the memory devices used in the Apple IIc and the way they are addressed. Input and output also use portions of the memory address space; refer to Chapter 2 for information.

Figure 11-10 illustrates the Apple IIc's overall memory bus organization and memory selection signals.

❖ *Note:* Some Apple IIc's have ROMs with 27xx designations, some have 23xx. They are functionally equivalent.

+5V	1	28	+5V
A12	2	27	(N.C.)
A7	3	26	A13
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	OE*
A2	8	21	A10
A1	9	20	CE*
A0	10	19	D7
D0	11	18	D6
D1	12	17	D5
D2	13	16	D4
GND	14	15	D3

**Figure 11-11**  
23128 ROM pinouts (In type  
23256 ROM, pin #27 is A14)



**Figure 11-10**  
Memory bus organization

## ROM addressing

In the Apple IIc the following programs are permanently stored in a type 23128 16K-by-8-bit ROM (Figure 11-11):

- ☐ Applesoft editor and interpreter
- ☐ Monitor
- ☐ enhanced video firmware

### UniDisk 3.5

The version of the Apple IIc that supports the UniDisk 3.5 uses a 23256 32K-by-8-bit ROM. It needs the extra space for the Protocol Converter, Mini-Assembler, and other added functions that it supports.

### Memory expansion

The Apple IIc that supports the memory expansion card also uses the 23256 ROM IC.

The ROM is enabled by two signals called *ROMEN1* and *ROMEN2*. (In the Apple IIc, ROMEN1 and ROMEN2 are electrically connected.) The segment of the ROM enabled by ROMEN1 occupies the memory address space from \$C100 through \$DFFF. The address space from \$C300 through \$C3FF and much of \$C800 through \$CFFF contains the enhanced video firmware.

These ROM address allocations are approximately true (some space sharing takes place):

- ☐ ROM addresses \$C000 through \$C0FF are never available.
- ☐ ROM addresses \$C100 and \$C200 are entry points to firmware for serial ports 1 and 2, respectively.
- ☐ ROM address \$C400 is the entry point to mouse interface support.
- ☐ ROM addresses \$C500 through \$C5FF are reserved.
- ☐ ROM address \$C600 is the entry point to firmware for the built-in and external disk drives. The built-in drive is considered slot 6 drive 1 or its equivalent. The external drive is considered slot 6 drive 2.
- ☐ ROM addresses starting at \$C700 support (from the Monitor) the external drive as if it were slot 7 drive 1, for external-drive startup only.
- ☐ Addresses \$D000 through \$F7FF contain the Applesoft BASIC interpreter; addresses \$F800 through \$FFFF contain the Monitor firmware.

KA7	1	24	+5V
KA6	2	23	KA8
KA5	3	22	CAPS
KA4	4	21	+5V
KA3	5	20	KBD*
KA2	6	19	LANGSW
KA1	7	18	GND
KA0	8	17	(N.C.)
D0	9	16	D6
D1	10	15	D5
D2	11	14	D4
GND	12	13	D3

**Figure 11-12**  
2316 ROM pinouts

### Memory expansion

+5V	1	28	+5V
A12	2	27	+5V
A7	3	26	+5V
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	GND
A2	8	21	A10
A1	9	20	WNDW*
A0	10	19	O7
O0	11	18	O6
O1	12	17	O5
O2	13	16	O4
GND	14	15	O3

**Figure 11-13**  
2364 pinouts

The Apple IIc that supports the memory expansion card has a ROM map that is different from that given for the original and UniDisk 3.5 IIc. The memory expansion ROM map is provided in Appendix I.

The other ROMs in the Apple IIc are a type 2316 ROM (Figure 11-12) used for the keyboard character decoder, and a type 2364 ROM (Figure 11-13) used for character sets for the video display. This 2364 ROM is rather large because it includes a section of straight-through bit-mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry.

---

## RAM addressing

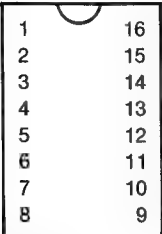
The RAM (programmable) memory in the Apple IIc is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 65C02 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIc, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIc take advantage of the two-phase system clock to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during  $\phi 0$ , and the display circuits read data only during  $\phi 1$ .

### Dynamic RAM refreshment

The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, the Apple IIc reads the data in the active display page and sends them to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIc refreshes the display 60 times per second.

The dynamic RAM devices used in the Apple IIc also need a kind of refreshment, because the data are stored in the form of electric charges that diminish with time and must be replenished. The Apple IIc is designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.



+5V	1	16	GND
MDx	2	15	CAS*
R/W*	3	14	MDx
RAS*	4	13	RA1
RA7	5	12	RA4
RA5	6	11	RA3
RA6	7	10	RA2
+5V	8	9	RA0

**Figure 11-14**  
64K RAM pinouts

The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called *multiplexing*. Because only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs (Figure 11-14).

## Memory expansion

In the Apple IIc that supports the memory expansion card, the 16 64Kx1 RAM ICs used for the original and UniDisk 3.5 IIc's are replaced by 4 64Kx4 ICs.

**Table 11-12**  
RAM address multiplexing

Mux'd address	Row address	Column address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

Different manufacturers' 64K RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described later in this chapter, the display circuitry generates a sequence of 8,192 memory addresses in high-resolution mode; in text and low-resolution modes, this sequence is the 1,024 display-page addresses repeated 8 times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every 2 milliseconds (see Table 11-12). This more than satisfies the refresh requirements of the dynamic RAMs.

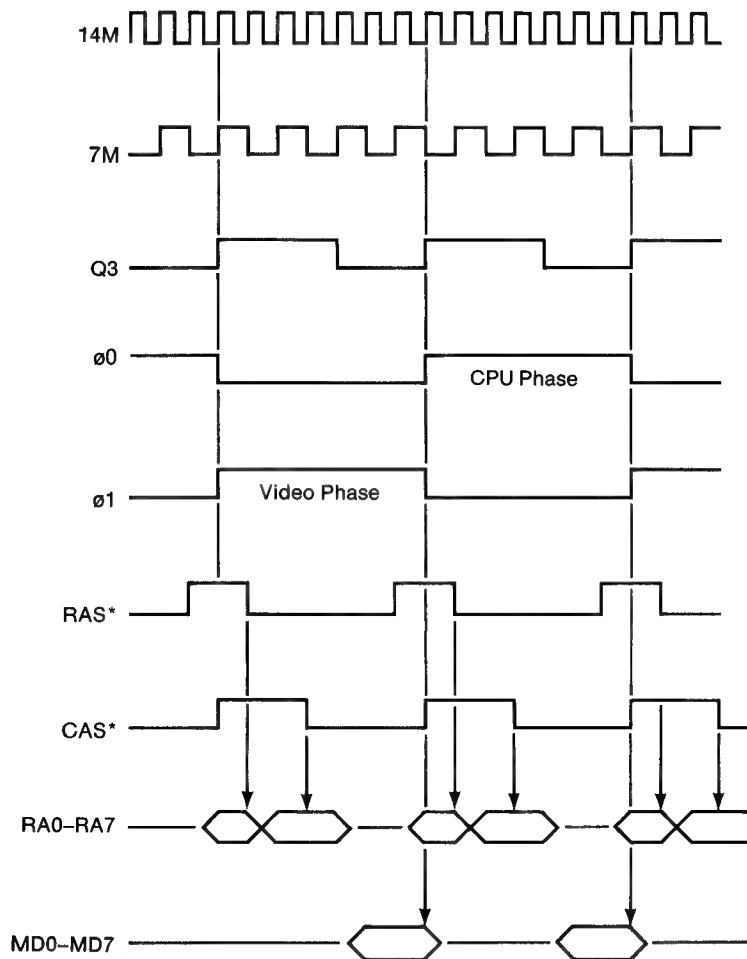
## Dynamic RAM timing

The Apple IIc's microprocessor clock runs at a speed of 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2-MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU are strobed by the falling edge of  $\phi 0$ , and display data are strobed by the falling edge of  $\phi 1$ , as shown in Figure 11-15.

The RAM timing looks complicated because the RAM address is multiplexed, as described previously. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines RA0–RA7 (Table 11-13). Along with the other timing signals, the TMG generates two signals that control the RAM addressing: row-address strobe (RAS) and column-address strobe (CAS).

**Table 11-13**  
RAM timing signals

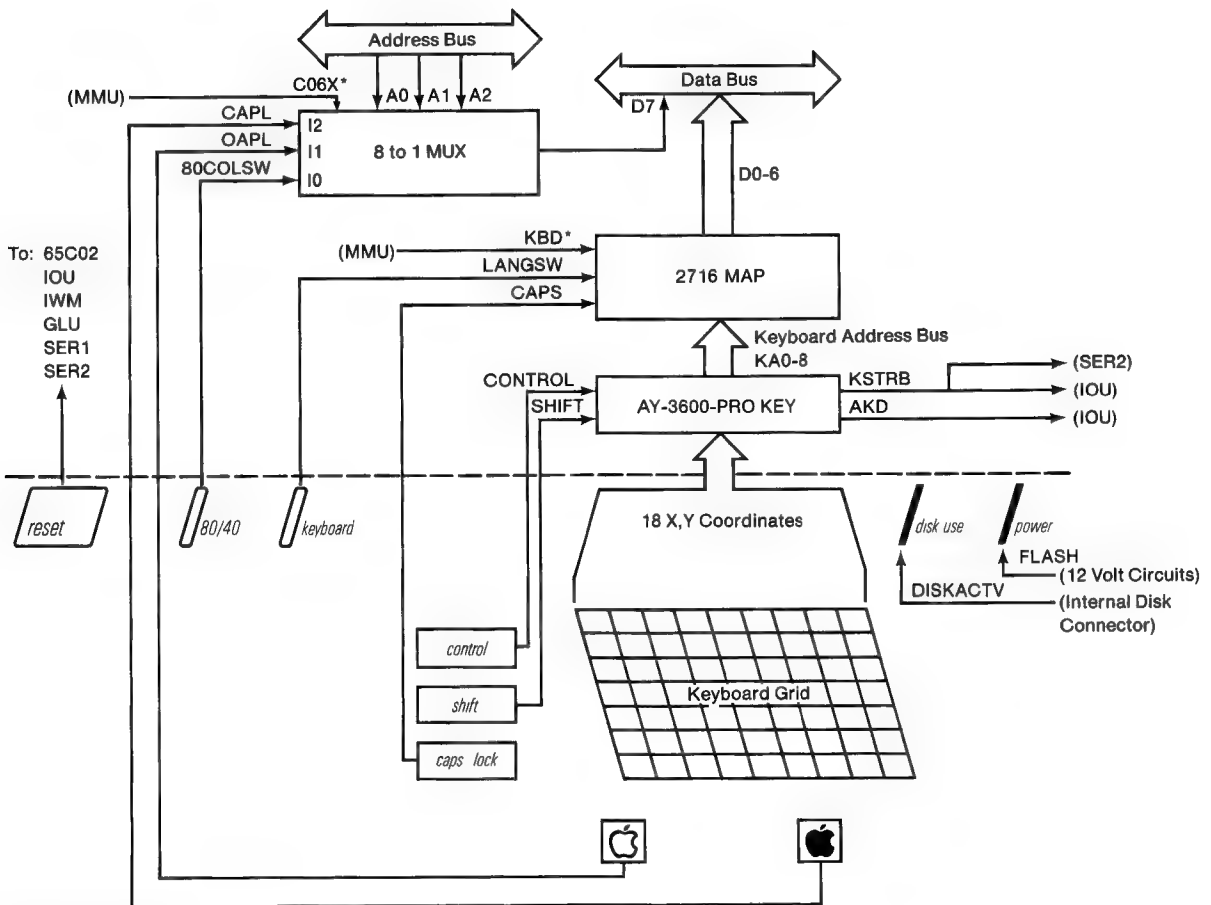
Signal	Description
$\phi 0$	Clock phase 0 (CPU phase)
$\phi 1$	Clock phase 1 (display phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternate RAM/column-address strobe
RA0-RA7	Multiplexed address bus
MD0-MD7	Internal data bus



**Figure 11-15**  
RAM timing signals

## The keyboard

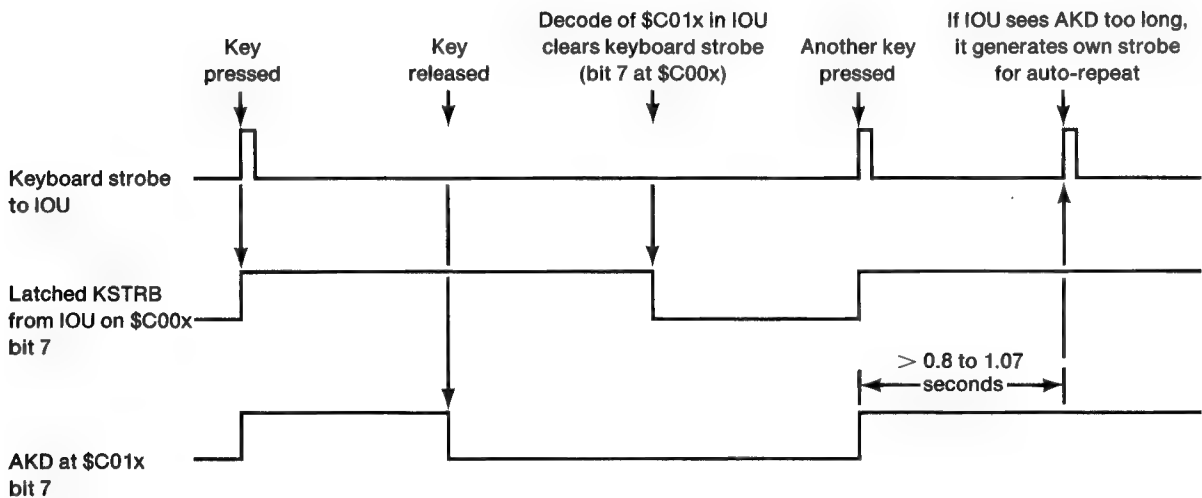
The Apple IIc's keyboard is a matrix of key switches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 26-pin connector (Figure 11-16). The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate is set by the external resistor-capacitor network made up of C46 and R6. The debounce time is also set externally, by C45.



**Figure 11-16**  
Keyboard circuit diagram

The AY-3600's outputs include five bits of key code plus separate lines for Control, Shift, any-key-down, and keyboard strobe. The any-key-down and keyboard-strobe lines are connected to the IOU, which addresses them as soft switches. The key-code line and Control and Shift are inputs to a separate 2316 ROM. The ROM translates them to the character codes that are enabled onto the data bus by signals named *KBD\** and *ENKBD\**. The KBD\* signal is enabled by the MMU whenever a program reads location \$C000, as described in Chapter 2.

Figure 11-17 illustrates the events that occur when a key is pressed, when the keypress is detected by a program, and when a key is pressed and held for more than about a second.

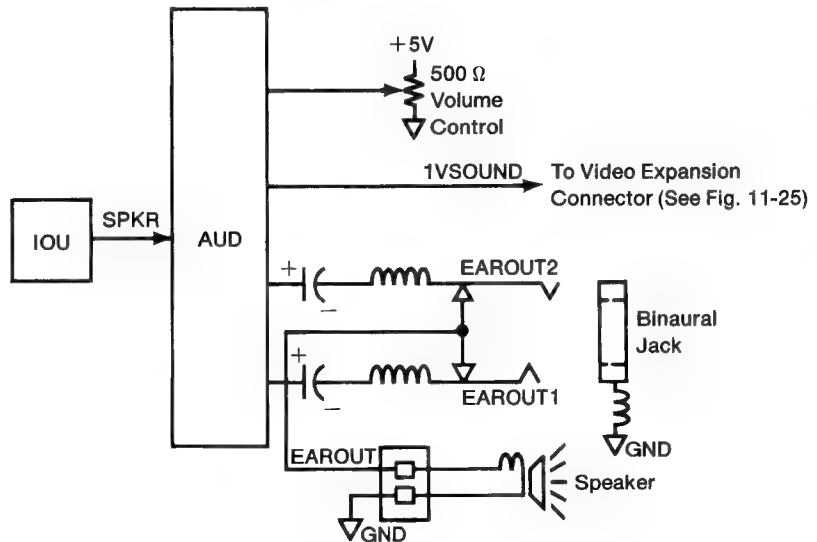


**Figure 11-17**  
Keyboard signals

---

## The speaker

The Apple IIc's built-in loudspeaker is controlled by a single bit of output from the input/output unit (IOU), amplified by a hybrid circuit (Figure 11-18).



**Figure 11-18**  
Speaker circuit diagram

**AUD** is an audio-amplifier hybrid circuit.

---

## Volume control

There is a 500-ohm variable resistor feeding anywhere from 0 to 5 volts to pin 5 of **AUD** to control the speaker volume. This potentiometer controls the volume of both the built-in speaker and whatever is plugged into the output jack.

---

## Output jack

Next to the volume control, along the lower-left side of the Apple IIc case, there is a 3.5-mm audio output jack. Although speaker output is monaural, the jack accommodates stereo headphone plugs (as well as monaural), providing sound to both channels. Inserting a headphone plug into the jack disconnects the built-in speaker.

---

---

## The video display

The Apple IIc produces a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that are being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

❖ *Note:* Apple IIc computers manufactured for sale in the USA generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIc's used in European countries require an external adapter to provide video that is compatible with the standard used there, which is called *PAL* (for phase alternating lines). References to the PAL standard are found in the bibliography at the end of this manual. This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a 1 corresponds to a voltage that generates a bright dot, and a 0 to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named *WNDW\** is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the *blanking intervals*, the display is blank and the *WNDW\** signal is high. The synchronization signals, called *sync* for short, are produced by making the signal named *SYNC\** low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

---

## The video counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE\*. The input to the horizontal counter is the 1-MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count from 0 to 64, then start over at 0. Whenever this happens, HPE\* forces another count with H0 through H5 held at 0, extending the total count to 65.

The IOU uses the 40 horizontal count values from 25 through 64 in generating the low-order part of the display data address. The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from 0. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 262 to generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIc's video display is not interlaced.)

---

## Display memory addressing

As described in Chapter 5, data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data are stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at hexadecimal \$0400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing them directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary.

The address transformation that folds three rows of 40 display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are described later in this chapter.

---

## Display address mapping

Consider the simplest display on the Apple IIc, the 40-column text mode. To address 40 columns requires 6 bits, and to address 24 rows requires another 5 bits, for a total of 11 address bits. Addressing the display this way would involve 2048 (2 to the 11th power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to display 1920 characters. The leftover chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- map the 960 bytes of 40-column text into only 1024 bytes
- scan the low-order address to refresh the dynamic RAMs
- continue to refresh the RAMs during video blanking

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly, and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals S0, S1, S2, and S3, where S stands for sum. Figure 11-19 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5\*. A constant value of 1 appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is 0 and the horizontal count is 24: H0, H1, H2, and H5 are 0's, and H3 and H4 are 1's. The value of the sum is 0, so the memory location for the first character on the display is the first location in the display page, as you might expect.

The requirements for RAM refreshing are discussed earlier in this chapter under "RAM addressing."

**Table 11-14**  
Display memory  
addressing

Memory address bit	Display address bit
A0	H0
A1	H1
A2	H2
A3	S0
A4	S1
A5	S2
A6	S3
A7	V0
A8	V1
A9	V2
A10	*
A11	*
A12	*
A13	*
A14	*
A15	GND

\* For these address bits,  
see Table 11-15.

			V3	Carry in
H5*	V3	H4	H3	Augend
V4	H5*	V4	1	Addend
S3	S2	S1	S0	Sum

**Figure 11-19**  
Display address transformation

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 11-14). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from 0 to 4 and the transformed address goes from 0 to 39, relative to the beginning of the display page.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 11-14, so that rows 0 through 7 start on 128-byte boundaries. When the vertical row counter reaches 8, V0, V1, and V2 are 0 again, and V3 changes to 1. If you do the addition in Figure 11-19 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

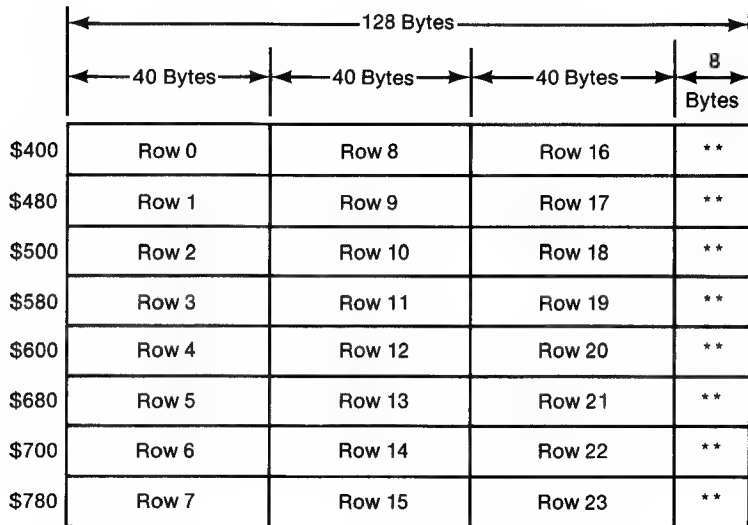
Table 11-14 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2, and S3 are the folded address bits described above. Table 11-15 shows memory address bits for the display modes.

**Table 11-15**  
Memory address bits for display modes

Address bit	Text and low resolution	High resolution and double high resolution
A10	80STORE+PAGE2'	VA
A11	80STORE'.PAGE2	VB
A12	0	VC
A13	0	80STORE+PAGE2'
A14	0	80STORE'.PAGE2

*Note:* Period (.) means logical AND; prime (') means logical NOT.

Figure 11-20 shows how groups of three 40-character rows are stored in blocks of 120 contiguous bytes starting on 128-byte address boundaries. This diagram is another way of describing the display mapping shown in Figure 5-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.



**Figure 11-20**  
40-column text display memory (memory locations marked with a double asterisk \*\* are screen holes, described in Chapter 2)

## Video display modes

The different display modes all use the address-mapping scheme described later in this chapter, but they use different-sized memory areas in different locations. This section describes the addressing schemes and the methods of generating the actual video signals for the different display modes. Figure 11-21 illustrates the video display circuits discussed in this section.



## Text displays

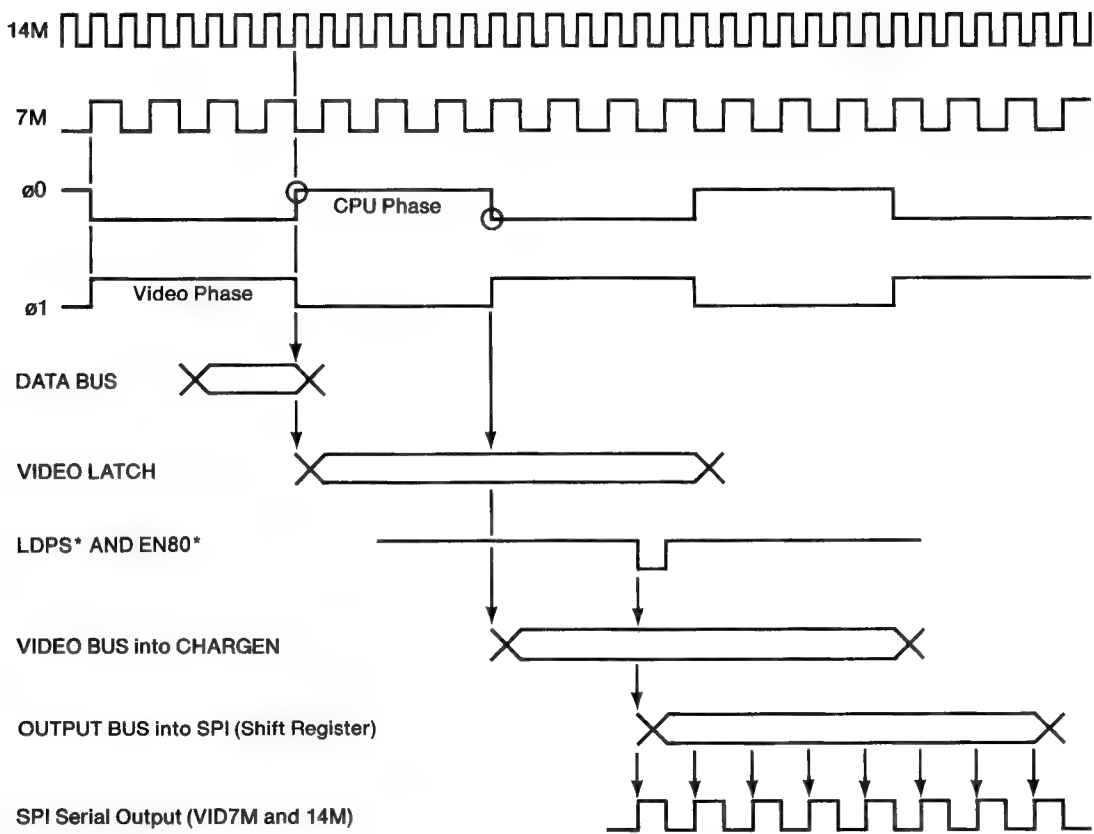
The text and low-resolution graphics pages begin at memory locations \$0400 and \$0800. Table 11-15 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by the settings of Page2 and 80Store, the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to 0. Notice that 80Store active inhibits Page2: there is only one display page in 80-column mode.

The low-order six bits of each data byte reach the character generator directly, via the video data bus VID0–VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator on lines RA9 and RA10.

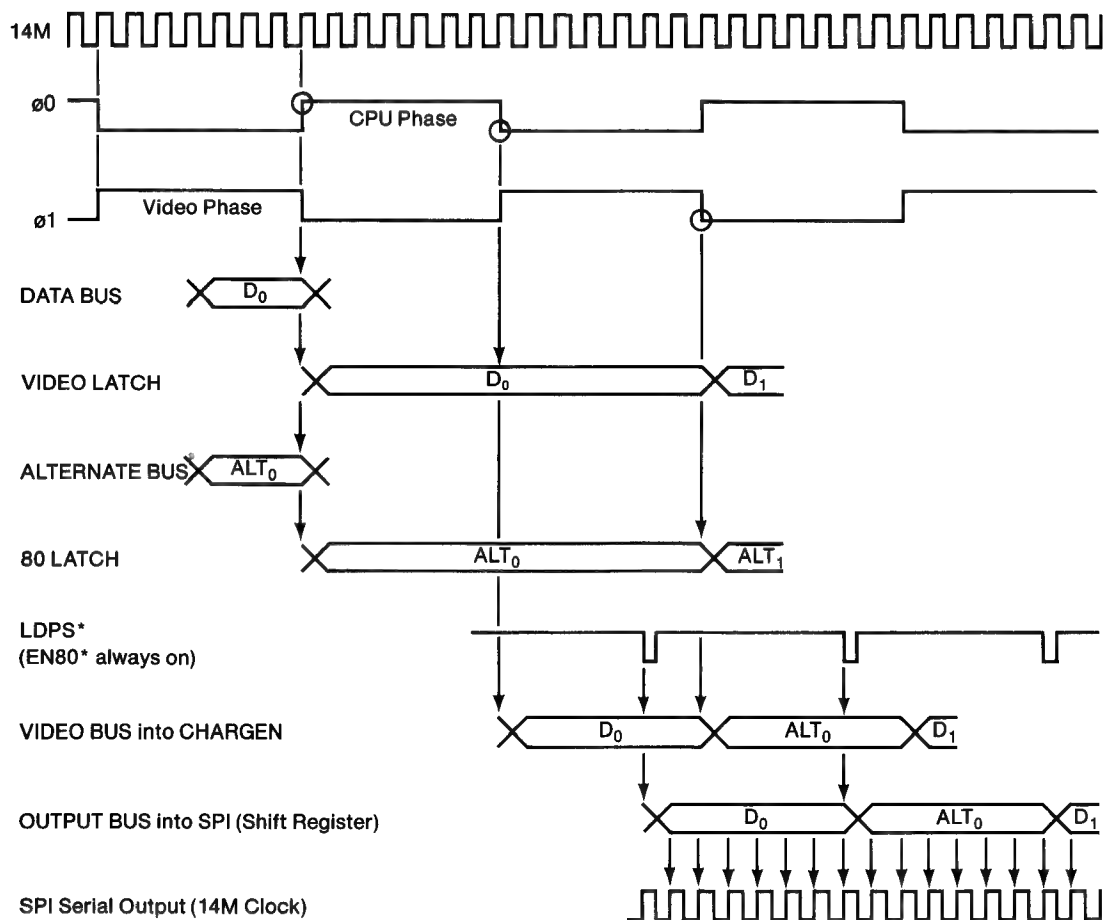
The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator puts out one of eight stored bit patterns selected by the three-bit number made up of VA, VB, and VC.

The bit patterns from the character generator are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit (Figure 11-21). The shift register is controlled by signals named LDPS\* (for load parallel-to-serial shifter) and VID7M (for video 7 MHz). In 40-column mode, LDPS\* strobes the output of the character generator into the shift register once each microsecond, and VID7M shifts the bits out at 7 MHz (Figure 11-22).

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory in auxiliary memory are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0–VID7) via separate 74LS374 three-state buffers. These buffers are loaded simultaneously (at the rising edge of  $\phi 0$ ), but their outputs are sent to the character generator alternately by the falling edge of  $\phi 0$  and  $\phi 1$ . In 80-column mode, LDPS\* loads data from the character generator into the shift register twice during each microsecond, once during  $\phi 0$  and once during  $\phi 1$ , and VID7M remains low, enabling the clock continuously at 14M (Figure 11-23).



**Figure 11-22**  
7-MHz video timing signals: 40-column, low-resolution, and high-resolution display



**Figure 11-23**  
 14-MHz video timing signals: 80-column and double high-resolution display

## Low-resolution display

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIRES\*, as shown in Table 11-16.

**Table 11-16**  
Character-generator control signals

Display mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIRES*	VC

The low-resolution graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in text mode, but each byte is interpreted as two nibbles. Each nibble selects 1 of 16 colors. During the upper four of the eight display lines, VC is low and the low-order nibble determines the color. During the lower four display lines, VC is high and the high-order nibble determines the color.

The bit patterns that produce the low-resolution colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits (Figure 11-21).

The video signal generated by the Apple IIc includes a short burst of 3.58-MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58-MHz color signal. The Apple IIc's video signal produces color by interacting with this 3.58-MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58-MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1-MHz data clock. To generate a stream of 14 bits from each 8-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same 8 bits; the last 2 bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: 1.02 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58-MHz color signal, the phase relationship between the bit patterns and the signal changes by a half cycle for each successive pattern. To compensate for this, the character generator puts out one of two different bit patterns for each nibble, depending on the state of H0, the low-order bit of the horizontal counter.

## High-resolution display

The high-resolution graphics pages begin at memory locations \$2000 and \$4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In high-resolution mode, these address bits are controlled by PAGE2 and 80STORE, the signals controlled by the display-page (Page2) and 80-column-video (80Col) soft switches. As in text mode, 80STORE inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In high-resolution graphics mode, the display data are still stored in blocks like the one shown in Figure 11-20, but there are eight of these blocks. As Tables 11-14 and 11-15 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display, VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block.

It might help to think of this scheme as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single high-resolution display, with each text display providing one line of dots in turn, instead of a row of characters.

The high-resolution bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the seven bits of display data. The low-order six bits of data reach the ROM via the video data bus VID0–VID5. The IOU sends the other two data bits to the ROM via RA9 and RA10.

The high-resolution colors described in Chapter 2 are produced by the interaction between the video signal the bit patterns generate and the 3.58-MHz color signal generated inside the monitor or TV set. The high-resolution bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58-MHz color signal. Any part of the video signal that produces a single white dot between two black dots, or vice versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating 1's and 0's gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating 1's and 0's.

To produce different colors, the bit patterns must have different phase relationships to the 3.58-MHz color signal. If alternating 1s and 0's produce a certain color, say green, then reversing the pattern to 0's and 1's will produce the complementary color, purple. As in the low-resolution mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the high-resolution software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58-MHz color signal. In high-resolution mode, the Apple IIc produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the TMG. If D7 is off, the TMG transmits shift-register timing signals LDPS\* and VID7M normally. If D7 is on, the TMG delays LDPS\* and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

- ♦ *A note about timing:* For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the TMG controls shift-register timing signals LDPS\* and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.

## Double high-resolution display

Double high-resolution graphics mode displays two bytes in the time normally required for one, but it uses high-resolution graphics Pages 1 and 1X instead of text and low-resolution Pages 1 and 1X.

❖ *Note:* There is a second pair of bytes, HRP2 and HRP2X, which can be used to display a second double high-resolution page.

Double high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, 7 from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appear in columns 0–6, 14–20, and so on, up to columns 547–552. Data from main memory appear in columns 7–13, 21–27, and so on, up to 553–559.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth (less than 14 MHz) monitor, single dots are dimmer than normal.

**RGB** stands for red, green, and blue, and identifies a type of color monitor that uses independent inputs for the three primary colors.

❖ *Note:* Except for some expensive **RGB**-type color monitors, any video monitor with a bandwidth as high as 14 MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

The main memory and auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The rising edge of  $\phi 0$  clocks a byte of main memory data into the video latch, and a byte of auxiliary memory data into the 80 latch (Figure 11-21).

$\Phi 1$  enables output from the (auxiliary) 80 latch, and  $\phi 0$  enables output from the (main) video latch. Output from both latches goes to CHARGEN, where GR and SEGB\* select high-resolution graphics. LDPS operates at 2 MHz in this mode, alternately gating the auxiliary byte and main byte into the parallel-to-serial shift register. VID7M is active (kept true) for double high-resolution display mode, so when it is ANDed with 14M, the result is still 14M. The 14M serial clock signal gates shift register output to VID, the video display hybrid circuit, for output to the display device.

For further information about double high-resolution graphics display, refer to the Bibliography.

**VID** is a video-amplifier hybrid circuit.

## Video output signals

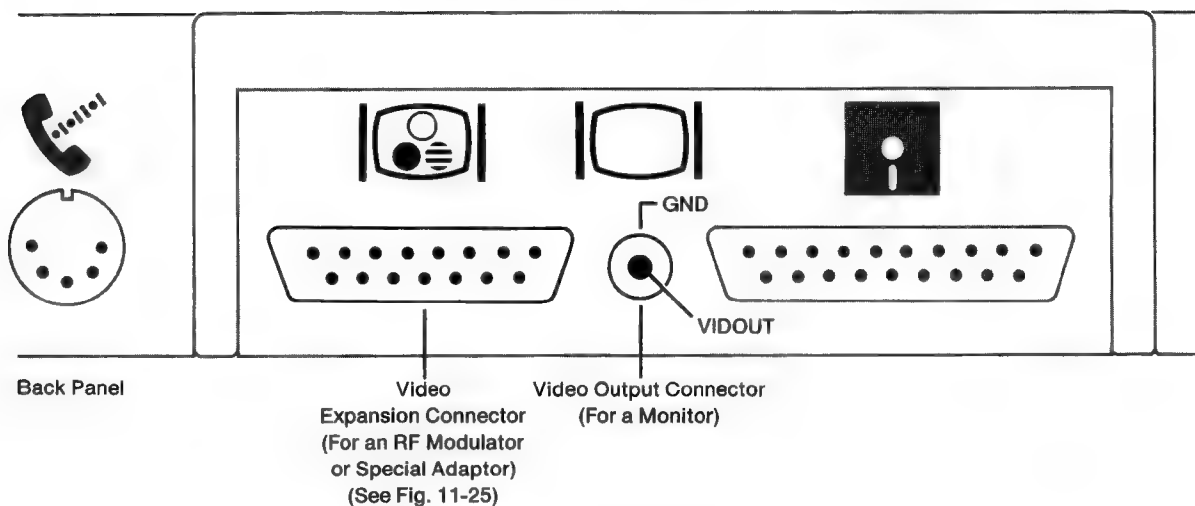
The stream of video data generated by the display circuits described above goes to a hybrid circuit (**VID**) that adjusts the signals to the proper amplitudes and conditions the color burst.

The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video. This signal is available in two places in the Apple IIc (Figure 11-24):

- at the video output connector on the back of the Apple IIc
- at the video expansion connector (pin 12) on the back panel (Table 11-17)

## Monitor output

The sleeve of the video output connector at the center of the Apple IIc back panel is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms. This arrangement is suitable for most video monitors.



**Figure 11-24**  
Video output back panel connectors

## Video expansion output

The back panel of the Apple IIc has a DB-15 connector for sophisticated video interfaces external to the computer. Figure 11-25 shows the pin assignments for this connector; Table 11-17 describes the signals. In Table 11-17, the column labeled *Deriv* indicates what clock signals the video signals are derived from. LDPS, CREF, and PRAS have a maximum delay of 30 ns from the appropriate 14-MHz rising edge. SEROUT is clocked out of a 74LS166 by the rising edge of 14M and has a maximum delay of 35 ns. VIDD7 is driven from a 74LS374 and has a maximum delay of 28 ns from the rising and (if 80-column) falling edges of  $\phi 1$ .

To align CREF so it is in the same phase at the beginning of every line, certain clock signals must be stretched. This stretch is for one 7M cycle (140 ns), and occurs at the end of each video line. All timing signals except 14M, 7M, and CREF are stretched.

---

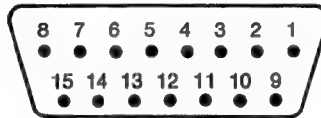
Warning	The maximum allowable current drain of +12V regulated power at the video expansion connector is 300 milliamps. If the external device draws more than this, it can damage the computer or cause the power supply to shut down.
---------	--

---

---

Warning	<p>The signals at the DB-15 on the Apple IIc are not the same as those at the DB-15 on the Apple III. Do not attempt to plug a cable intended for one into the other.</p> <p>Several of these signals, such as 14 MHz, must be buffered within about 4 inches (10 cm) of the back panel connector—preferably inside a container directly connected to the back panel. For technical information, contact Apple Technical Support.</p>
---------	---

---



Pin	Signal	Pin	Signal
1	TEXT	9	PRAS*
2	14M	10	GR
3	SYNC*	11	SEROUT*
4	SEGB	12	NTSC
5	1VSOUND	13	GND
6	LDPS*	14	VIDD7
7	WNDW*	15	CREF
8	+12V		

**Figure 11-25**  
Video expansion connector pinouts

**Table 11-17**  
Video expansion connector signals

Pin	Deriv	Signal	Description
1	ø0	TEXT	Video text signal from TMG; set to inverse of GR, except in double high-resolution mode
2		14M	14-MHz master timing signal from the system oscillator
3	Q3	SYNC*	Displays horizontal and vertical synchronization signal from IOU pin 39
4	PRAS	SEGB	Displays vertical counter bit from IOU pin 4; in text mode indicates second low-order vertical counter; in graphics mode indicates low-resolution
5		1VSOUND	One-volt sound signal from pin 5 of the audio hybrid circuit (AUD)
6	14M	LDPS*	Video shift-register load enable from pin 12 of TMG
7	PRAS	WNDW*	Active area display blanking; includes both horizontal and vertical blanking

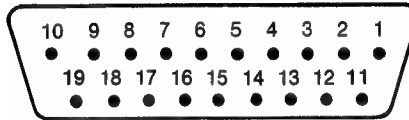
**Table 11-17 (continued)**  
Video expansion connector signals

Pin	Deriv	Signal	Description
8		+12V	Regulated +12 volts DC; can drive 300 mA
9	14M	PRAS*	RAM row-address strobe from TMG pin 19
10	PRAS	GR	Graphics mode enable from IOU pin 2
11	14M	SEROUT*	Serialized character-generator output from pin 1 of the 74LS166 shift register
12		NTSC	Composite NTSC video signal from VID hybrid chip
13		GND	Ground reference and supply
14	ø0	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	14M	CREF	Color reference signal from TMG pin 3; 3.58 MHz

## Disk I/O

Disk I/O—for both the built-in and the external drive—is supported by the IWM disk controller unit. The external drive is attached via a DB-19 connector. Figure 11-26 shows this connector. Table 11-18 describes the pin assignments. Supply voltages come from the power supply; all other signals come from the IWM, described earlier in this chapter.

**Warning** The power available at this connector is for a Disk IIc or similar drive only. Do not use power from the external disk connector for any other purpose—you may damage the internal voltage converter. To derive external power for an attached device, use one of the other connectors and observe the current limits given in this manual.



Pin	Signal	Pin	Signal
1,2,3,4	GND	13	SEEKPH2
5	-12V	14	SEEKPH3
6	+5V	15	WRREQ*
7,8	+12V	16	N.C.
9	EXTINT*	17	DR2*
10	WRPROT	18	RDDATA
11	SEEKPH0	19	WRDATA
12	SEEKPH1		

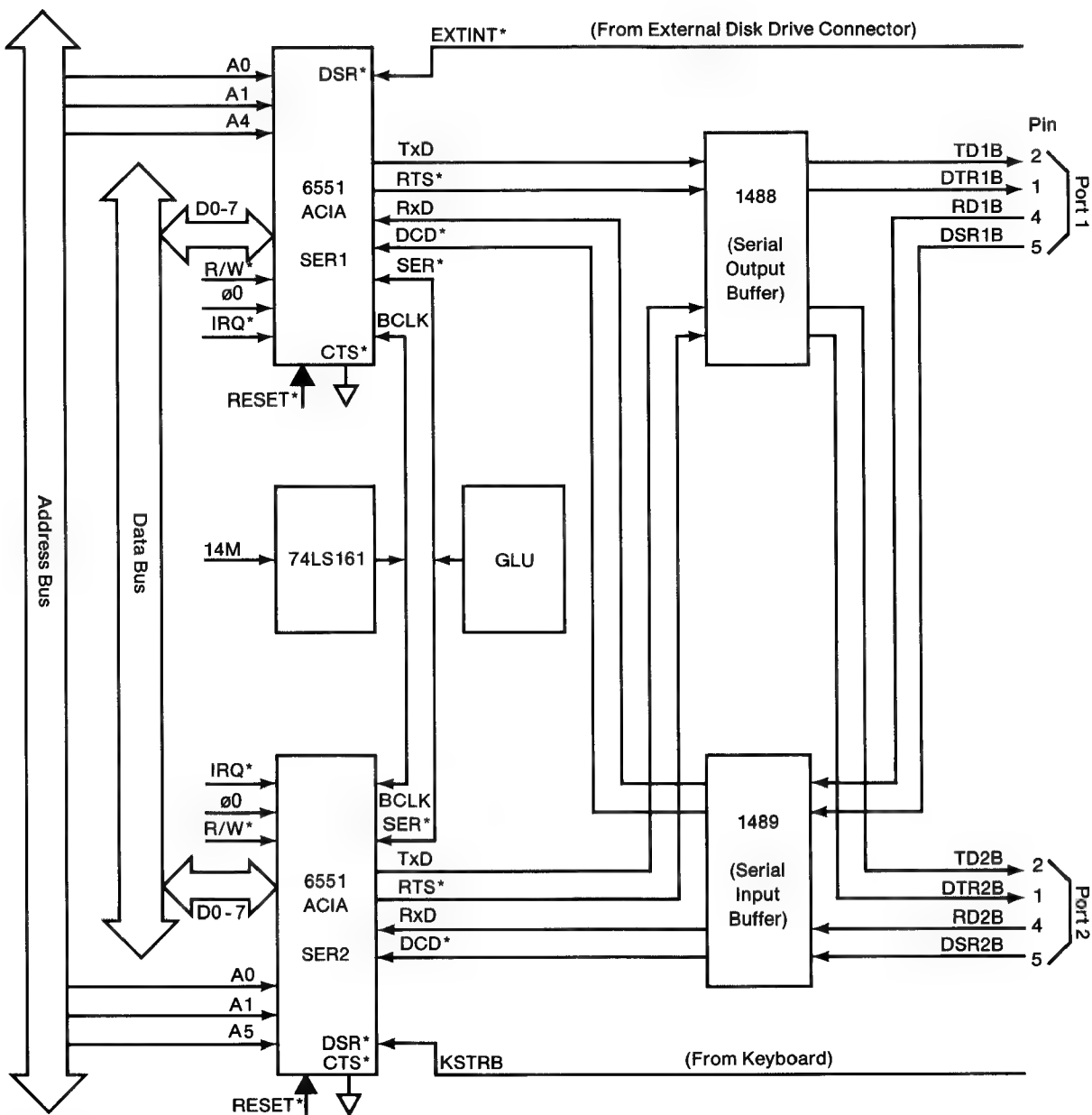
**Figure 11-26**  
Disk drive connector

**Table 11-18**  
Disk drive connector signals

Connector pin	Signal	Description
1,2,3,4	GND	Ground reference and supply
6	+5V	+5 volt supply
7,8	+12	+12 volt supply
9	EXTINT*	External interrupt
10	WRPROT	Write-protect input
11-14	ø0-4	Motor phase 0-4 output
15	WRREQ*	Write request
17	DR1*	Drive 1 select
18	RDDATA	Read data input
19	WRDATA	Write data output

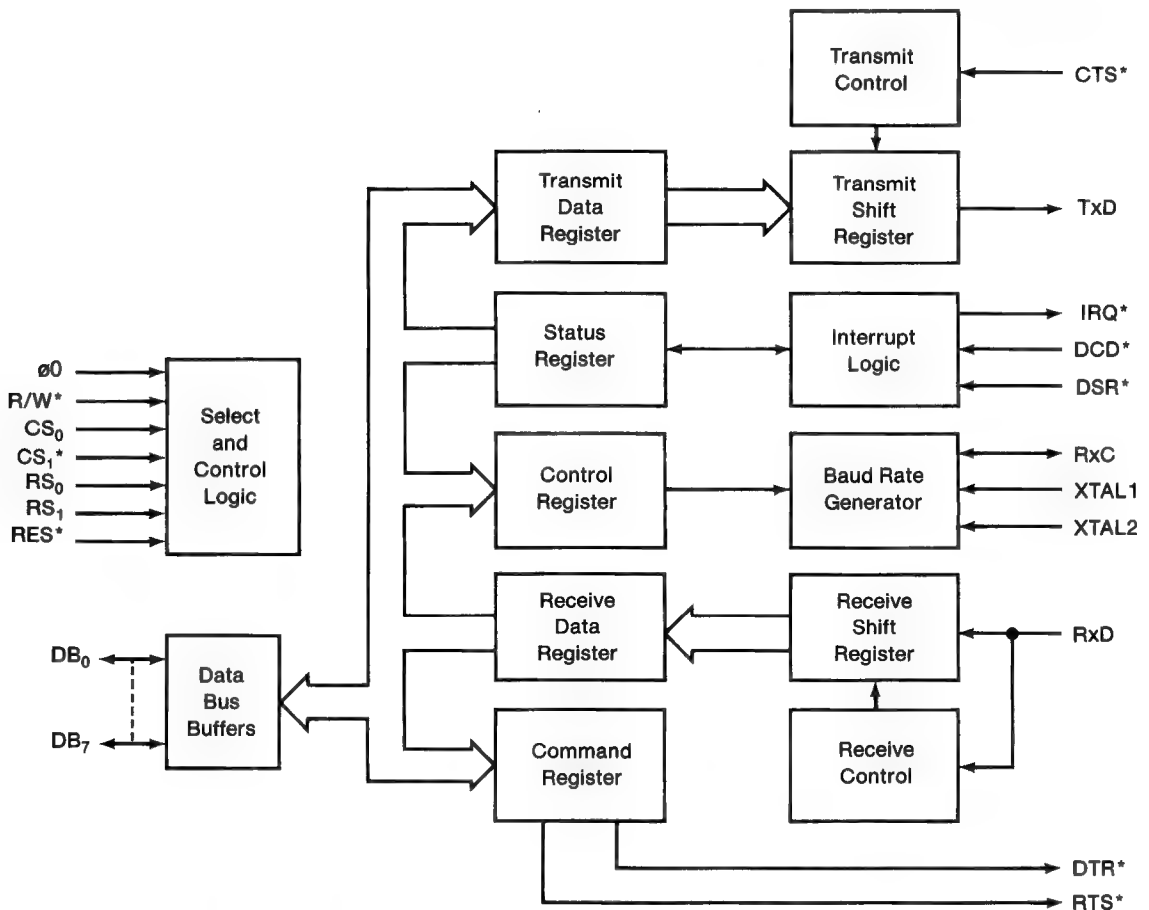
## Serial I/O

The Apple IIc has built into it two 6551 asynchronous communication interface adapters (ACIA) and supporting input and output buffers for full-duplex serial communication. Figure 11-27 is a block diagram of the Apple IIc serial ports. ACIA outputs are buffered by a 1448-quad line driver. Similarly, ACIA inputs are buffered by a 1489-quad line receiver.



**Figure 11-27**  
Serial port circuits

Figure 11-28 is a detailed block diagram of the 6551 ACIA. The registers are described later in this chapter.



**Figure 11-28**  
6551 ACIA block diagram (copyright © 1978 by Synertek Inc.; used by permission)

The 6551 pin assignments are shown in Figure 11-29 and described in Table 11-19. Note that the two 6551's are not used in exactly the same way—each one supports a different set of interrupts.

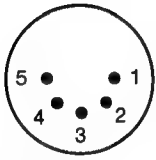
Port 1 reads external interrupts (EXTINT\*) on its Data Set Ready (DSR) pin. This input is tied to +5V through a 3.3-K $\Omega$  pullup resistor.

GND	1	28	R/W*
A5	2	27	ø0
SER*	3	26	IRQ*
RESET*	4	25	D7
(N.C.)	5	24	D6
BCLK	6	23	D5
(N.C.)	7	22	D4
RTS*	8	21	D3
CTS*	9	20	D2
TxD	10	19	D1
(N.C.)	11	18	D0
RxD	12	17	DSR*
A0	13	16	DCD*
A1	14	15	+5V

**Figure 11-29**  
6551 pinouts

**Table 11-19**  
6551 signal descriptions

Pin	Signal	Description
1	GND	Power and signal common ground
2	A4 A5	Address line 4 to select serial port 1 Address line 5 to select serial port 2
3	SER*	Serial device select from GLU
4	RESET*	Resets both serial ports
5	N.C.	Not connected
6	BCLK	Baud rate clock from GLU
7	N.C.	Not connected
8	RTS*	Request to Send output
9	CTS*	Clear to Send input (not used on IIC; tied to ground)
10	TXD	Transmit Data output
11	N.C.	Not connected
12	RXD	Receive Data input
13,14	A0,A1	Address lines 0 and 1
15	+5V	+5 volt supply
16	DSR	DCD* pin; used on IIC as Data Set Ready input
17	EXTINT* KSTRB	DSR*pin; used on IIC as External interrupt (port 1 ACIA), or Keyboard strobe input (port 2 ACIA; Appendix E)
18–25	D0–D7	8-bit data bus
26	IRQ*	Interrupt Request input
27	ø0	Phase 0 clock pulse
28	R/W*	Read/write select input



Pin	Port 1	Port 2
1	DTR1B	DTR2B
2	TD1B	TD2B
3	GND	GND
4	RD1B	RD2B
5	DSR1B	DSR2B

**Figure 11-30**  
Serial port connectors

The back panel connectors for both serial ports are 5-pin DIN jacks. The pin assignments are shown in Figure 11-30 and described in Table 11-20.

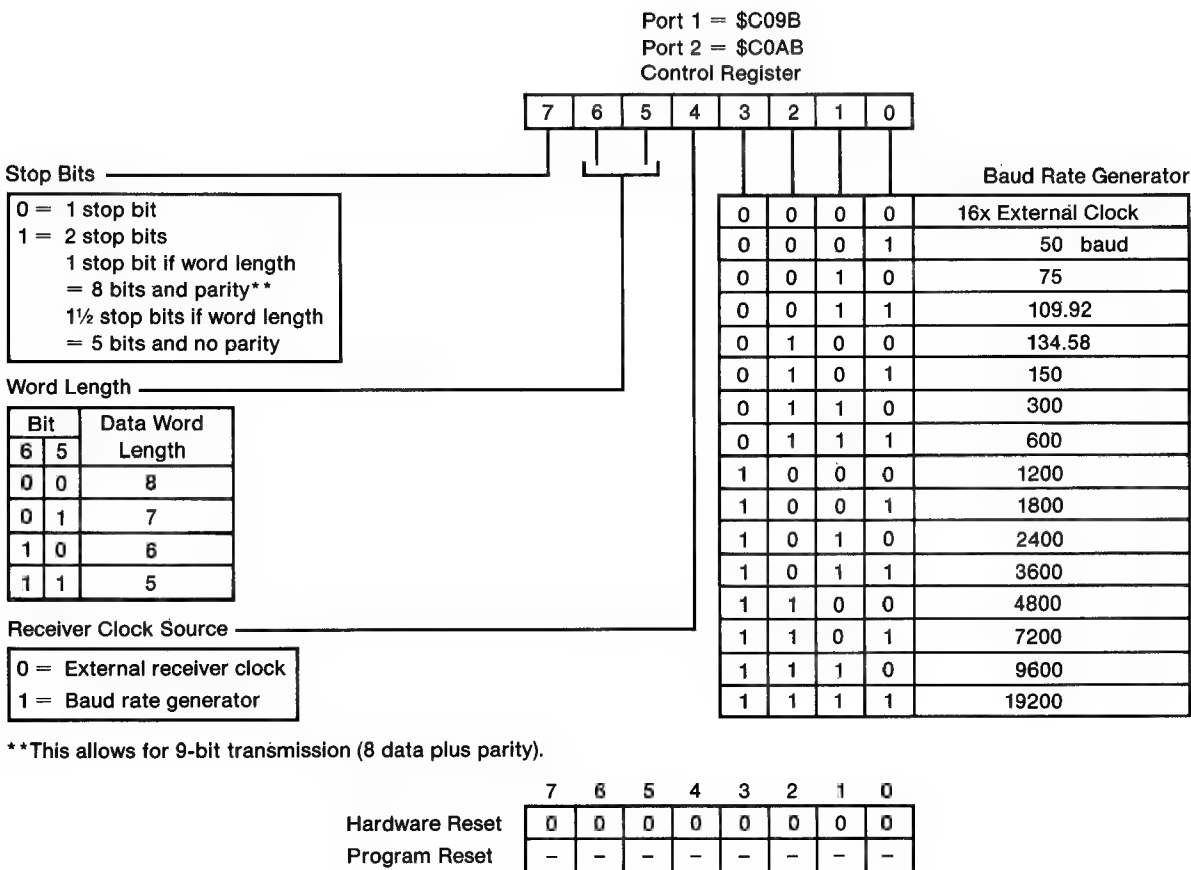
**Table 11-20**  
Serial port connector signals

Pin	Signal	Description
1	DTR1B DTR2B	Data Terminal Ready output
2	TD1B TD2B	Transmit Data output
3	GND	Power and signal common
4	RD1B RD2B	Read Data input
5	DSR1B DSR2B	Data Set Ready input

## ACIA control register

Figure 11-31 shows the bit assignments for the ACIA control register, which the hardware locates at address \$C09B for serial port 1, and \$C0AB for serial port 2. This register determines the number of data and stop bits the ACIA will transmit and receive, and the clock source and baud rate to use for data transfer.

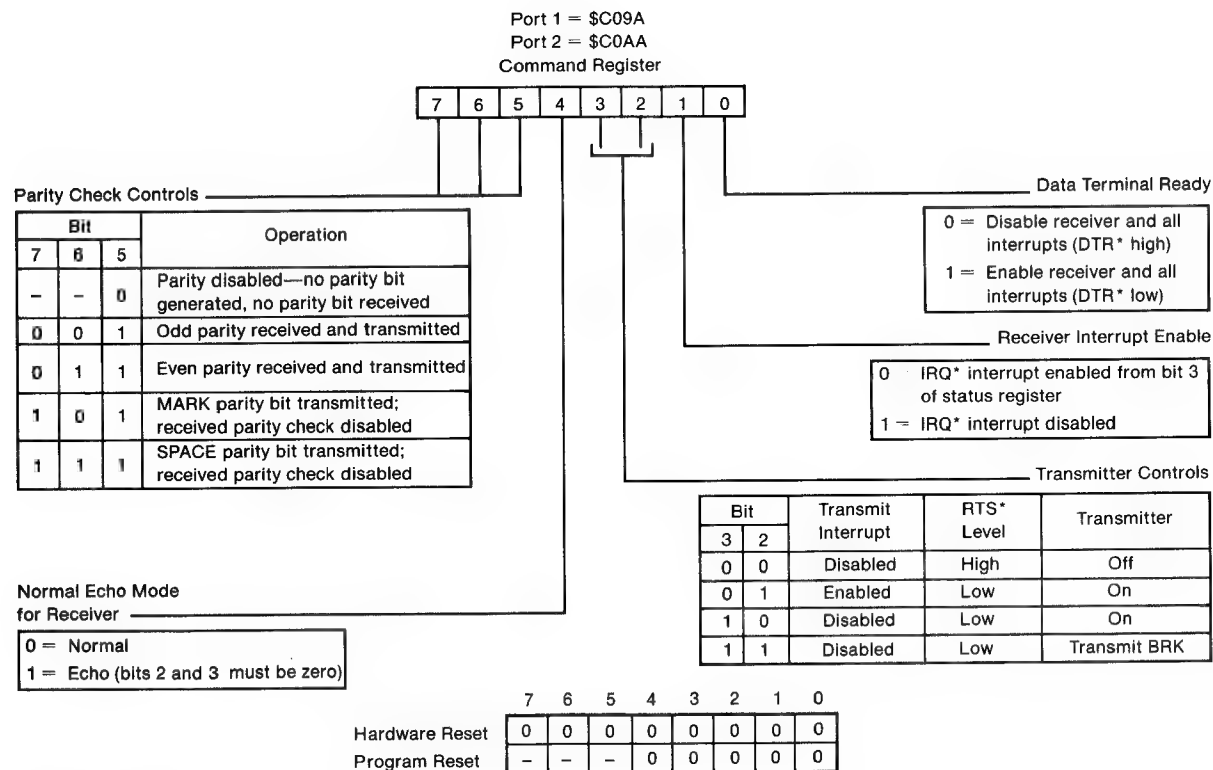
The receiver clock source is derived from the Apple IIc's TMG chip; the resulting baud rates are equal to or up to two percent lower than the nominal rate. (The EIA standard allows plus or minus two percent variation.) If an Apple IIc serial port is used with a modem that is two percent above the nominal rate, framing errors can occur, especially at 1200 baud and above, when using 8 data bits. It may be necessary to select a lower baud rate for 8-bit binary data transfers.



**Figure 11-31**  
ACIA control register (copyright © 1978 by Synertek Inc.; used by permission)

# ACIA command register

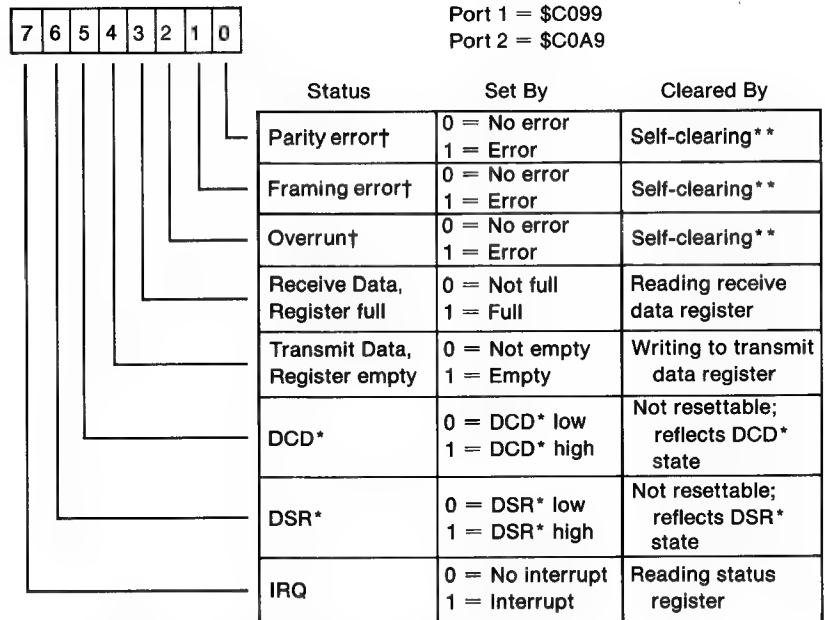
Figure 11-32 shows the bit assignments for the ACIA command register, which the hardware locates at address \$C09A for serial port 1, and at \$C0AA for serial port 2. This register controls specific transmit and receive functions: parity checking, echoing input to output, allowing transmit and receive interrupts, and setting levels for Data Terminal Ready and Request to Send.



**Figure 11-32**  
ACIA command register (copyright © 1978 by Synertek Inc.; used by permission)

## ACIA status register

Figure 11-33 shows the bit assignments for the ACIA status register, which is hard-wired to address \$C099 for serial port 1, and \$C0A9 for serial port 2. This register reports the condition of the transmit/receive register, errors detected during data transfer, and the level of the Data Carrier Detect, Data Set Ready, and Interrupt Request lines.



† No interrupt generated for these conditions.

\*\* Cleared automatically after a read of RDR and the next error-free receipt of data.

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	-	-	-	-	-	-	-	-

**Figure 11-33**

ACIA status register (copyright © 1978 by Synertek Inc.; used by permission)

---

## ACIA transmit/receive register

Each ACIA uses the same address—\$C098 for serial port 1, \$C0A8 for serial port 2—as temporary storage for both transmission and reception of data.

When the register is used for transmitting data, bit 0 is the leading bit to be transmitted; unused data bits are the high-order bits, which are ignored.

When the register is used for receiving data, bit 0 is the first bit received; unused data bits are the high-order bits, which are set to 0. Parity bits never appear in the receive data register; they are stripped off after being used for external parity checking.

---

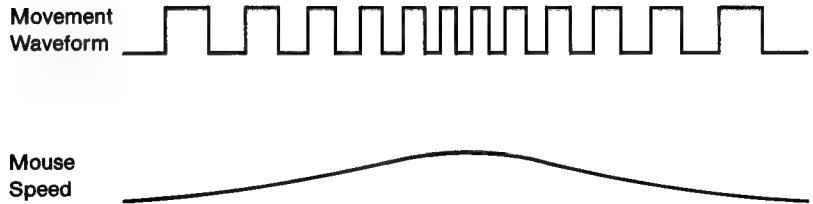
---

## Mouse input

The mouse is a hand-held X-Y pointing device that can be rolled along a flat surface. It has an attached pushbutton. This section describes how mouse movement and direction can be detected and interpreted.

A mouse has a ball inside its housing that protrudes a small distance so that its turning corresponds to mouse movements across a table top. Two wheels inside the housing, set at 90-degree angles to each other, follow movements of the ball; this causes two disks to rotate. The disks have rectangular holes arranged near their edges, making them resemble circular slide mounts used with stereoscopic slide viewers.

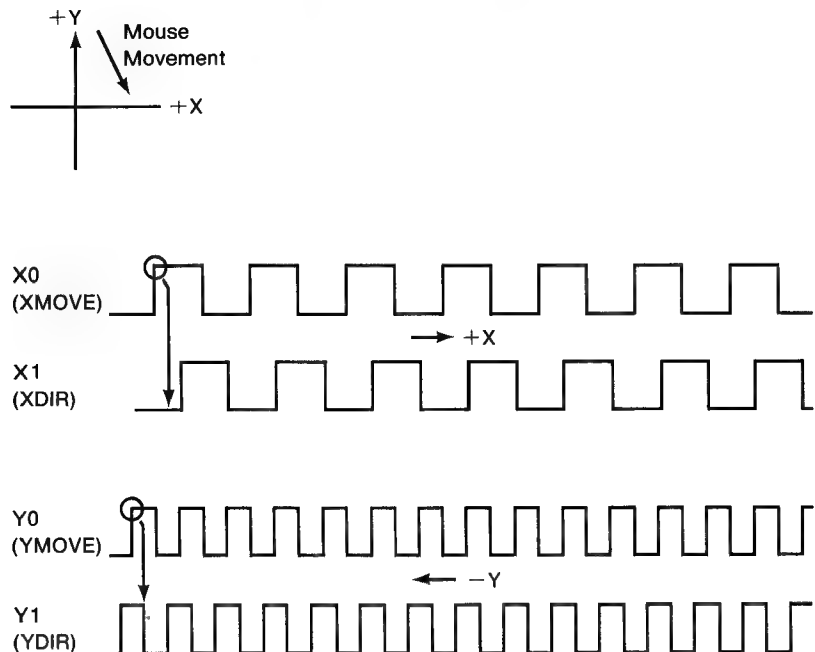
The light from a tiny infrared emitter reaches a photoreceptor whenever one of the holes on the disk lies between them. An internal circuit in the mouse causes the resulting voltage to swing quickly to a 1 or a 0 value as soon as a certain threshold is crossed. The result is something approximating a square wave (Figure 11-34) that varies directly with the speed of mouse movement. One of these indicates the X component (X0) of mouse movement; the other, the Y component (Y0).



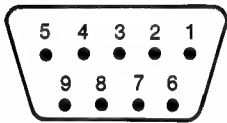
**Figure 11-34**  
Sample mouse waveform

Under program control, either the rising edge or the falling edge of each square wave can cause an interrupt, which the firmware handles by updating a counter. However, the program needs to know whether to add or to subtract 1 from a counter; that is, it needs to know the direction of X or Y movement.

There is a second infrared emitter/photoreceptor pair almost 180 degrees opposite the first pair for each disk. These pairs are positioned in such a way that the square waves they generate are approximately a quarter-wave offset from their respective movement waves (Figure 11-35). These waveforms are called *X1* (X direction) and *Y1* (Y direction).



**Figure 11-35**  
Mouse movement and direction waveforms



Pin	Signal
1	MOUSEID*
2	+5V
3	GND
4	XDIR
5	XMOVE
6	(N.C.)
7	MSW*
8	YDIR
9	YMOVE

**Figure 11-36**  
Mouse connector

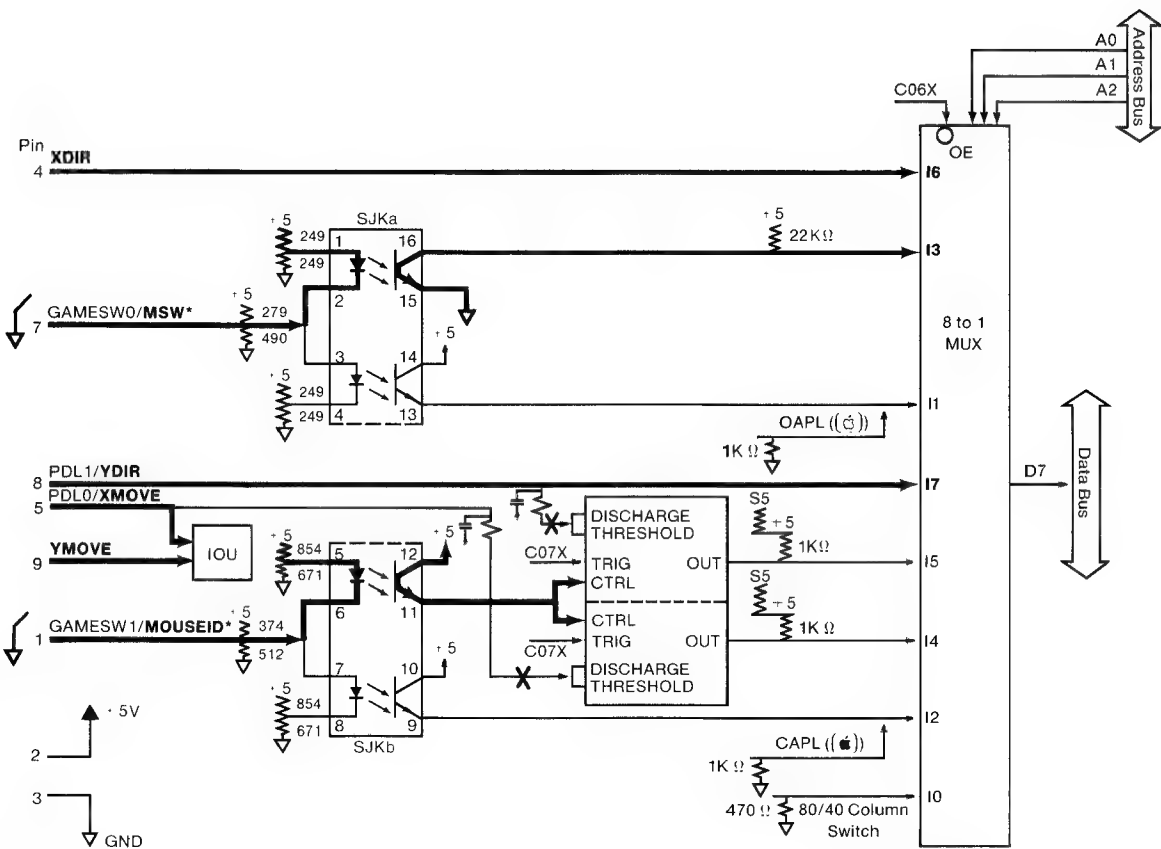
When a rising edge of X0 causes an interrupt, a mouse-driver program can immediately check whether X1 is 0 (indicating a movement to the right) or 1 (indicating a movement to the left). Similarly, the mouse driver can read Y1 immediately after a Y0 interrupt to determine whether the mouse moved up or down one count along the Y axis.

Figure 11-36 shows the pin assignments for the mouse DB-9 connector on the back panel. Table 11-21 gives the signal names and descriptions.

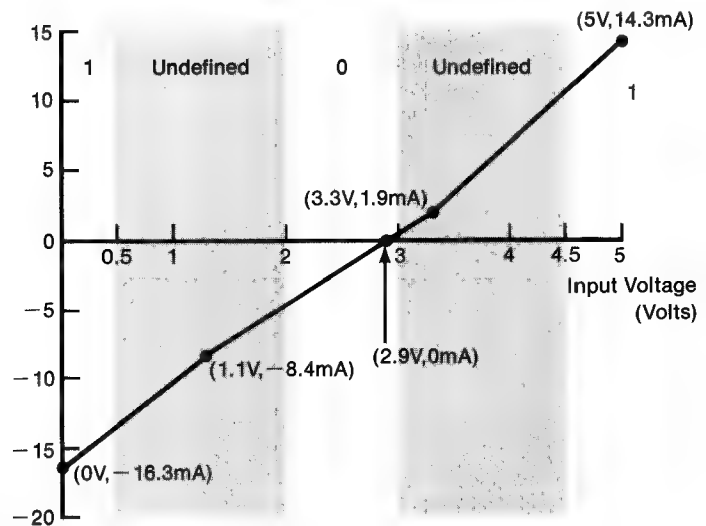
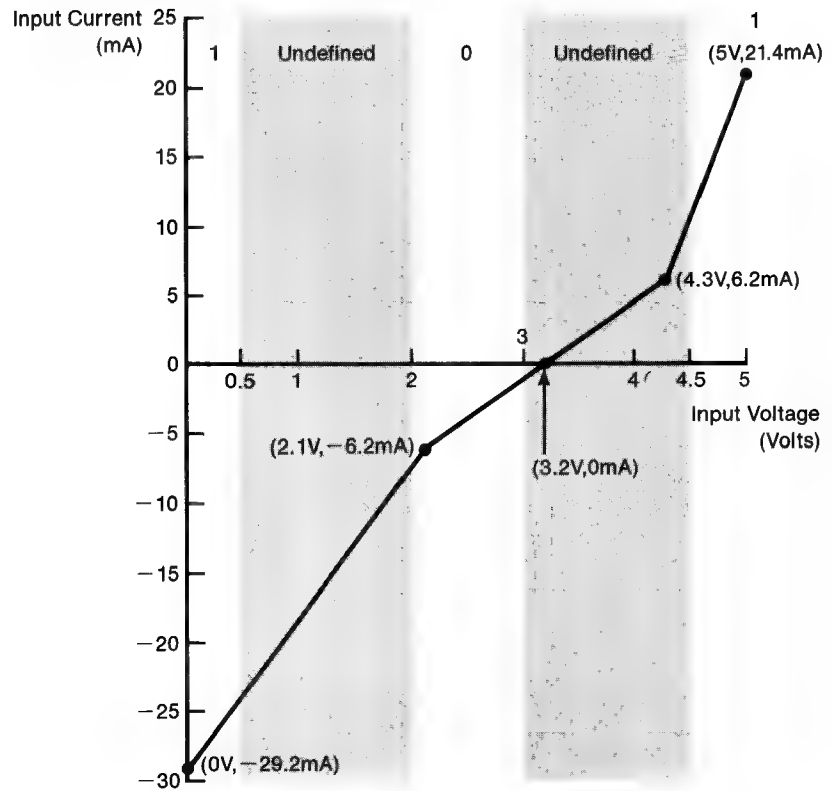
**Table 11-21**  
Mouse connector signals

Pin	Signal	Description
1	MOUSEID*	Mouse identifier: when active, disables NE556 hand controller timer
2	+5V	Total current drain from this pin must not exceed 100 mA
3	GND	System ground
4	XDIR	Mouse X-direction indicator
5	XMOVE	Mouse X-movement interrupt
6	N.C.	Not connected
7	MSW*	Mouse button
8	YDIR	Mouse Y-direction indicator
9	YMOVE	Mouse Y-movement interrupt

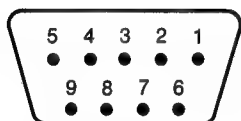
Figure 11-37 shows the mouse and hand controller circuitry with the mouse circuits emphasized. Figure 11-38 illustrates the values of the mouse-button circuit when the button is pressed or not pressed. Pressing the button disables the NE556 by pulling the reset comparator threshold value up so that it cannot reset the flip flop. As a result the mouse-button input value remains at a TTL level.



**Figure 11-37**  
Mouse circuits



**Figure 11-38**  
Mouse button signals



Pin	Signal
1	GAMESW1
2	+5V
3	GND
4	Not used for hand controllers
5	PDL0
6	(N.C.)
7	GAMESW0
8	PDL1
9	Not used for hand controllers

**Figure 11-39**  
Hand controller connector

## Hand controller input

Several input signals that are individually controlled via soft switches are collectively referred to as the **hand controller** (game) signals. These signals arrive in the Apple IIc via the same DB-9 connector as the one used for the mouse, but the Apple IIc interprets these signals differently.

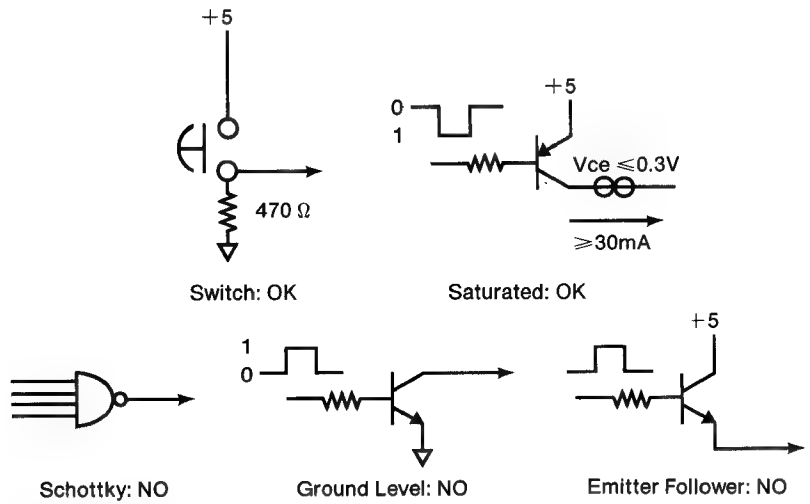
The DB-9 connector pin assignments and signal descriptions, as used for hand controller input, appear in Figure 11-39 and Table 11-22.

Even though they are normally used for hand controllers, these signals can be used for other simple I/O applications. There are two 1-bit switch inputs, labeled *Sw0* and *Sw1*, and two analog inputs, called *paddles* and labeled *Pdl0* and *Pdl1*. Figure 11-40 shows how to connect the 1-bit switch inputs for compatibility with all other Apple II series computers.

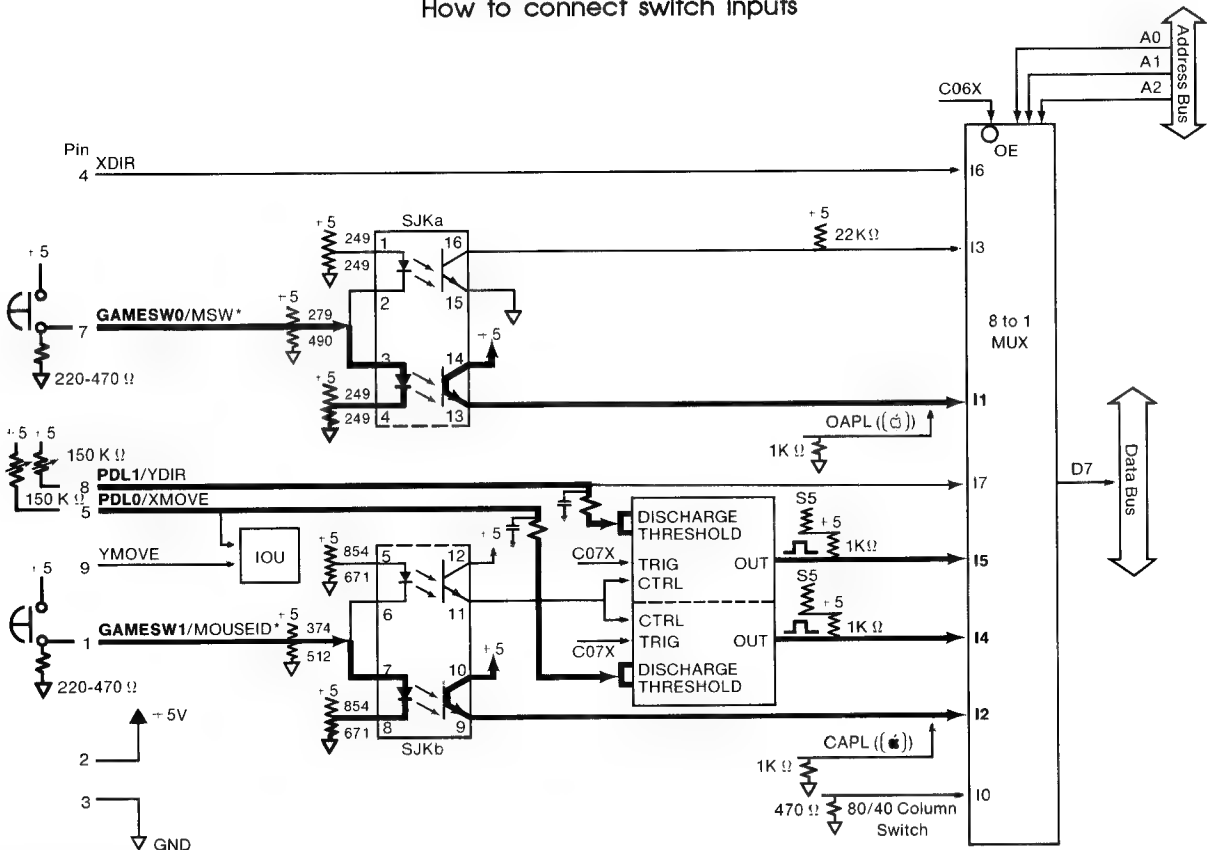
The switch inputs are multiplexed by a 74LS251 8-to-1 multiplexer enabled by the C06X\* signal from the MMU. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus. Figure 11-41 shows the mouse and hand controller circuitry with the hand controller circuits highlighted. Figure 11-42 illustrates the values of the hand controller switch inputs when the switch is open or closed.

**Table 11-22**  
Hand controller connector signals

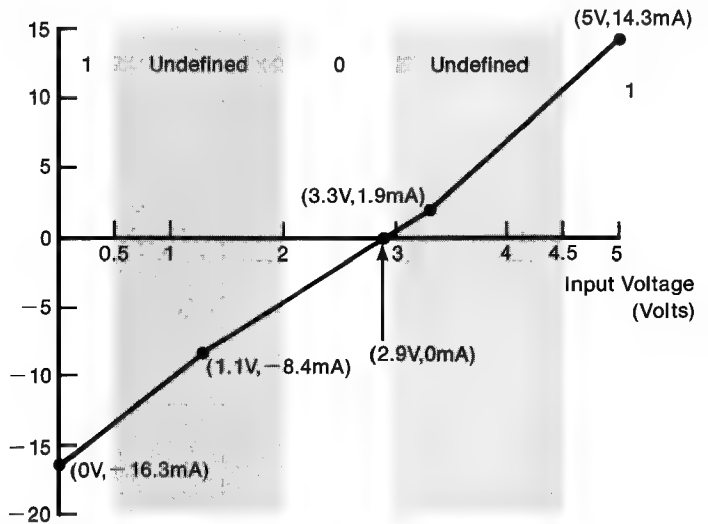
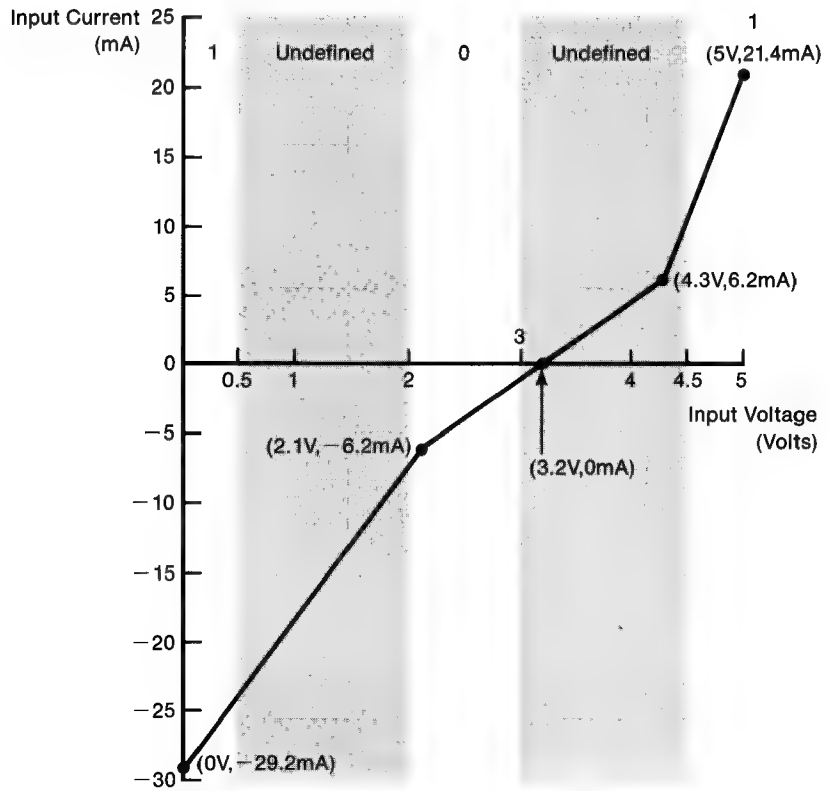
Pin	Signal	Description
1	GAMESW1	Switch input 1 (sometimes called <i>paddle button 1</i> ).
2	+5V	+5V power supply; total current drain from this pin must not exceed 100 mA.
3	GND	System ground.
4,9		Not used for hand controllers.
5,8	PDL0 and PDL1	Hand controller inputs; each of these must be connected to a 150-K $\Omega$ variable resistor connected to +5V.
6	N.C.	Not connected.
7	GAMESW0	Switch input 0 (sometimes called <i>paddle button 0</i> ).



**Figure 11-40**  
How to connect switch inputs



**Figure 11-41**  
Hand controller circuits



**Figure 11-42**  
Hand controller signals

The hand controller inputs are connected to the timing inputs of an NE556 dual analog timer. Addressing \$C07X sends a signal from MMU pin 22 that resets both timers and causes their outputs to go to 1 (high). A variable resistance of up to 150 K $\Omega$  connected between one of these inputs and the +5V supply controls the charging time of one of the two 0.022 microfarad capacitors.

When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

---

**Warning**

The only way to ensure correct paddle values is to make sure the output of the paddle you intend to read is low before you trigger the timer. Triggering the timer starts the charging cycle for the capacitor in each paddle circuit; the cycle for one may not be completed by the time you have read the other. If you retrigger or read the other paddle too soon (that is, in less than 3 ms), you will get a false value for it.

---

---



---

## Memory expansion card

Memory expansion card I/O is supported by an internal connector mounted on the main logic board. Figure 11-43 is a pinout diagram for this connector.

For information on the Apple IIc Memory Expansion Card, refer to the *Apple IIc Memory Expansion Card Reference*.

	Pin	Signal	Pin	Signal
2 ● ● 1	1	D0	18	A9
4 ● ● 3	2	D1	19	A10
6 ● ● 5	3	D2	20	A11
8 ● ● 7	4	D3	21	A12
10 ● ● 9	5	D4	22	A13
12 ● ● 11	6	D5	23	A15
14 ● ● 13	7	D6	24	A15
16 ● ● 15	8	D7	25	RESET
18 ● ● 17	9	GND	26	RW
20 ● ● 19	10	GND	27	+5V
22 ● ● 21	11	A0	28	+5V
24 ● ● 23	12	A1	29	PHO
26 ● ● 25	13	A4	30	GND
28 ● ● 27	14	A5	31	7M
30 ● ● 29	15	A6	32	GND
32 ● ● 31	16	A7	33	Q3
34 ● ● 33	17	A8	34	+5V

**Figure 11-43**  
Memory expansion card connector pinout diagram

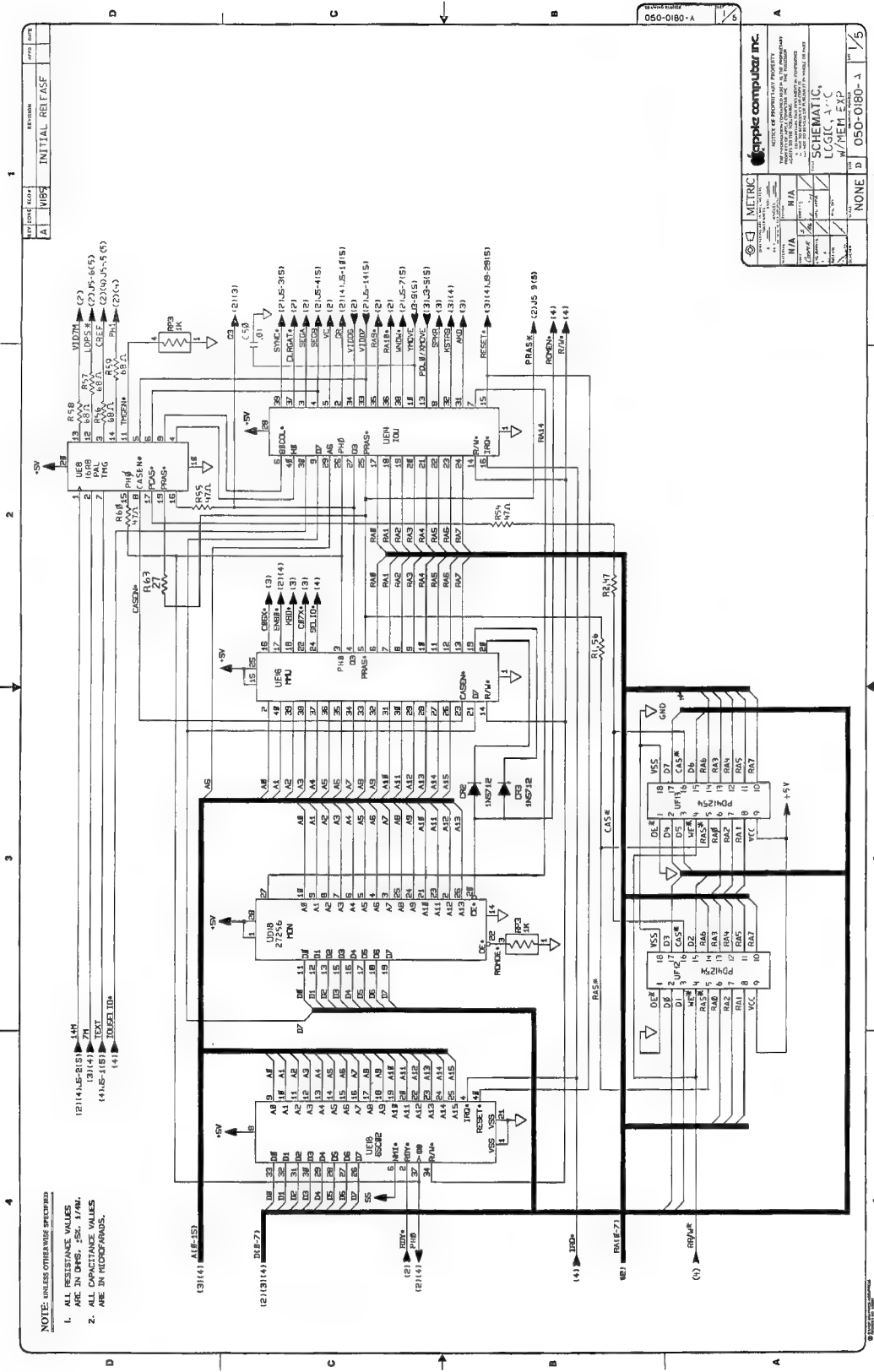
---



---

## Schematic diagrams

Figure 11-44, on the following pages, is a set of schematic diagrams for the Apple IIc.



**Figure 11-44a**  
Apple IIc schematic diagram, part 1



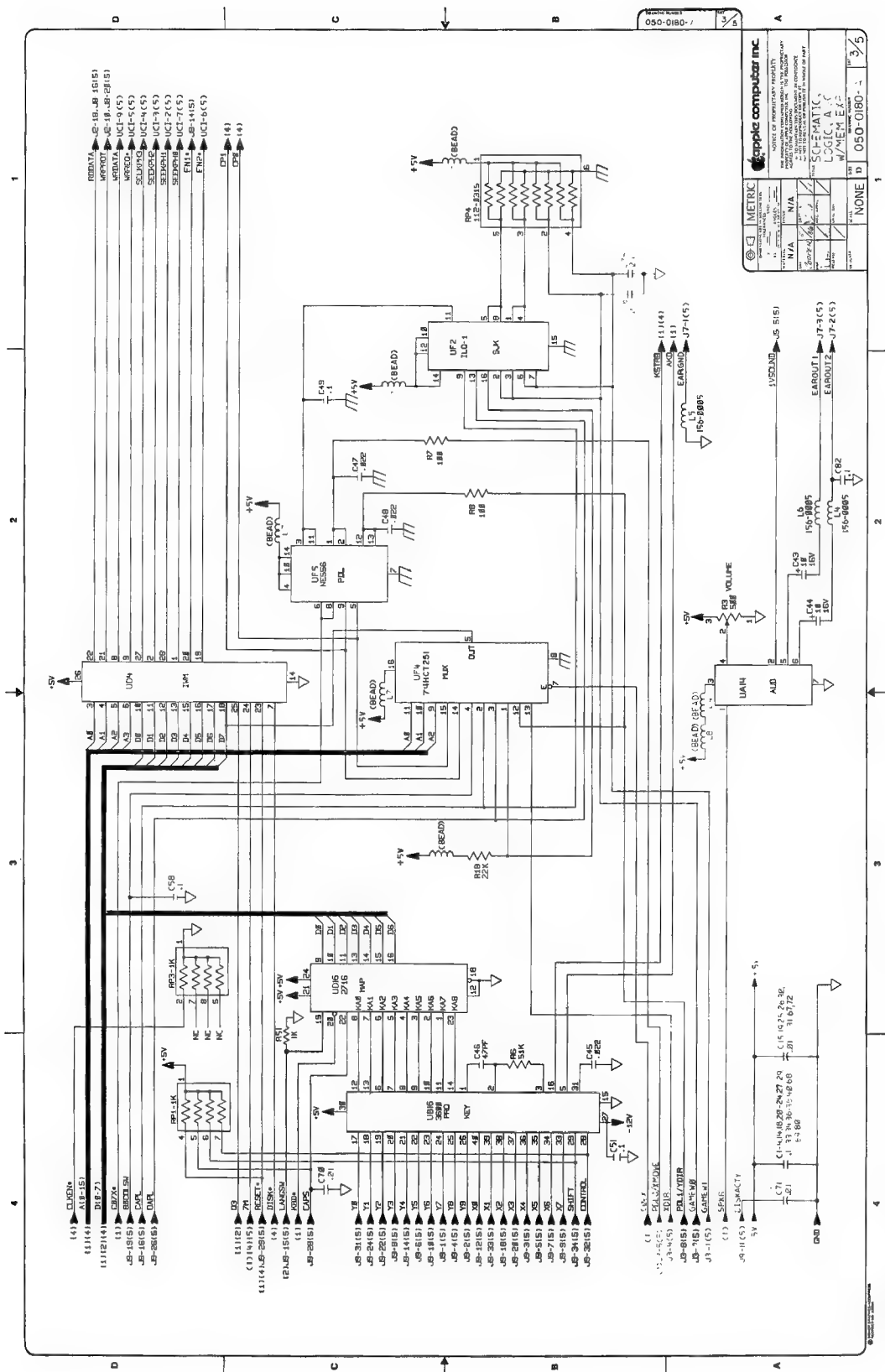
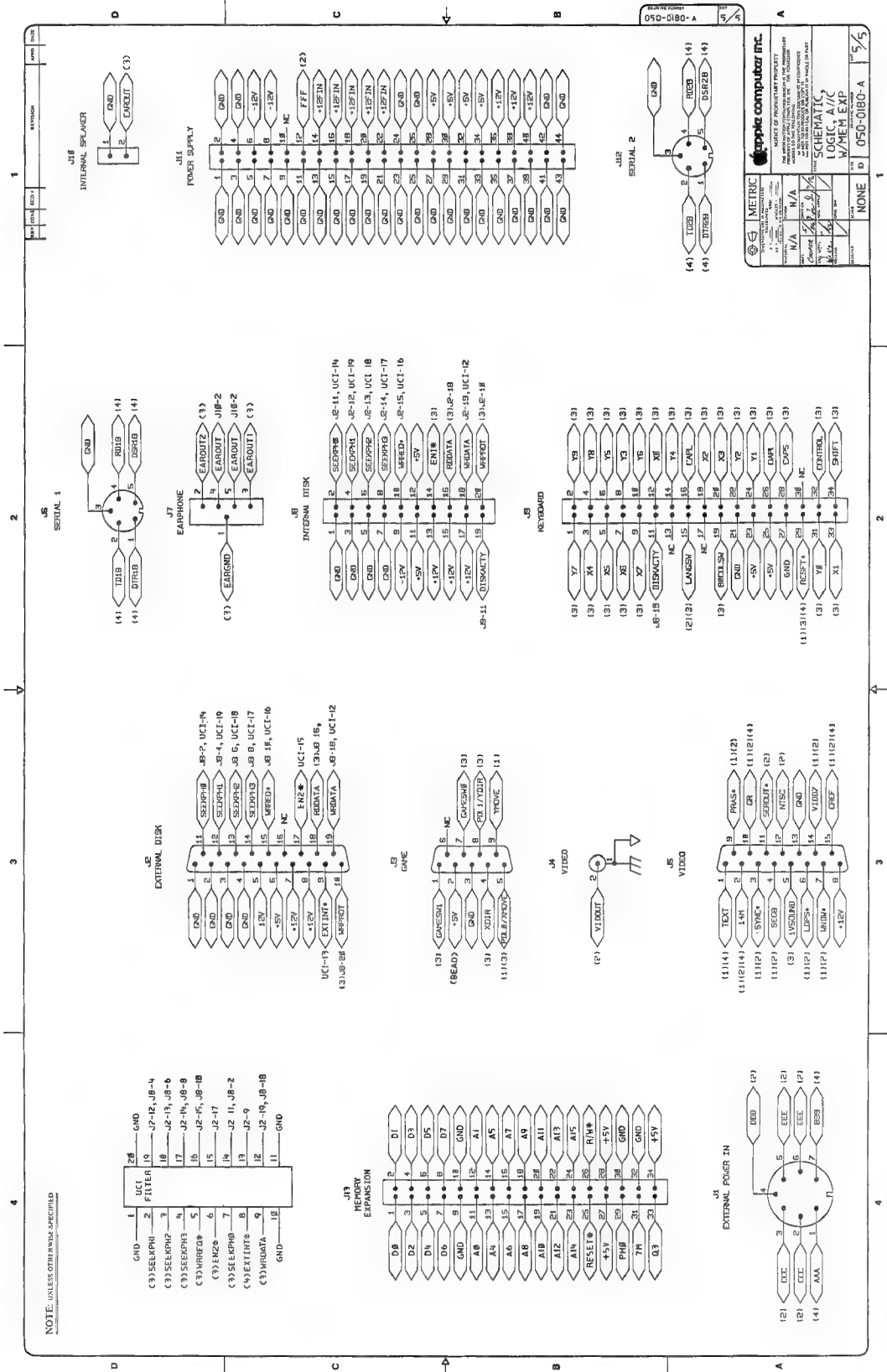


Figure 11-44c  
Apple IIC schematic diagram, part 3





**Figure 11-44e**  
Apple IIc schematic diagram, part 5



## Appendix A



# The 65C02 Microprocessor

This appendix describes the differences between the 6502 and the 65C02 microprocessors. It also contains the data sheet for the NCR 65C02 microprocessor.

In the data sheet tables, execution times are specified in numbers of cycles. One cycle for the Apple IIc equals 0.978 microseconds.

If you want to write programs that execute on all computers in the Apple II series, make sure your code uses only the subset of 65C02 instructions present on the 6502.

---

---

### Differences between 6502 and 65C02

The data sheet in this chapter lists the new 65C02 instructions and addressing modes. This section supplements that information by listing the instructions whose execution times or results have changed from their 6502 counterparts.

---

#### Differing cycle times

In general, differences in execution times are significant only in time-dependent code, such as precise wait loops. Fortunately, instructions with changed execution times are few.

Table A-1 lists the 65C02 instructions whose number of instruction execution cycles is different from their number on the 6502.

**Table A-1**  
Cycle time differences

Instruction/mode	Opcode	6502 cycles	65C02 cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

## Differing instruction results

The instructions that have different results from their 6502 equivalents are

- BIT (in immediate mode)
- JMP (indirect, when crossing a page boundary).

The BIT instruction when used in immediate mode (code \$89) leaves processor status register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the status register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6. However, all BIT instructions on both versions of the processor set status bit 1 (Z) if the memory location being tested contains a 0.

If the JMP indirect instruction (code \$6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP (\$02FF) gets ADL from location \$02FF on both processors. On the 65C02, ADH comes from \$0300 while on the 6502, ADH comes from \$0200.

## Data sheet

The rest of this appendix is copyright 1982, NCR Corporation, Dayton, Ohio, and is reprinted with their permission.

## GENERAL DESCRIPTION

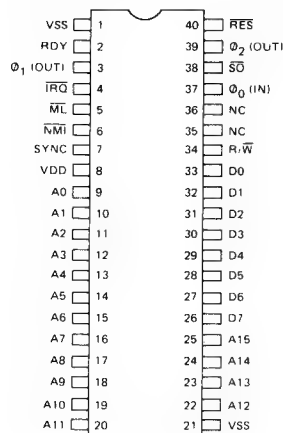
The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

## FEATURES

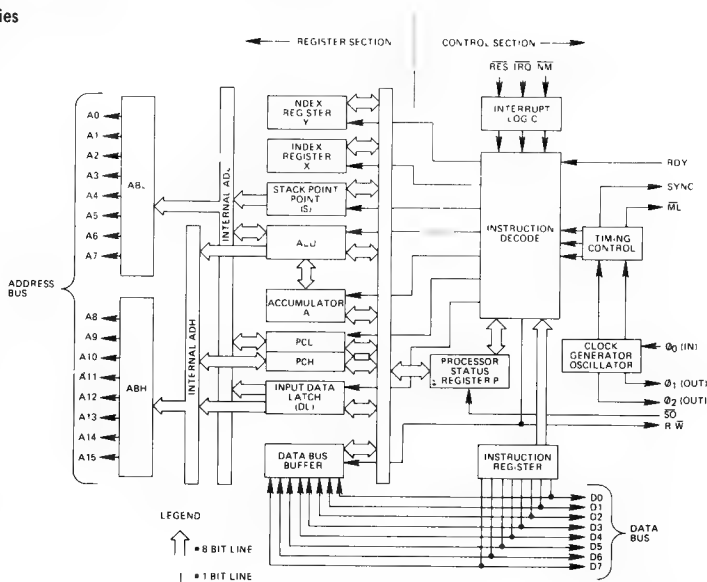
- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 KHz for even lower power consumption (pseudo-static: stop during  $\Phi_2$  high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for (RDY, IRQ,  $\overline{SO}$ , NMI and RES)

\* Specifications are subject to change without notice.

## PIN CONFIGURATION



## NCR65C02 BLOCK DIAGRAM



## NCR65C02

### ■ ABSOLUTE MAXIMUM RATINGS: ( $V_{DD} = 5.0 \text{ V} \pm 5\%$ , $V_{SS} = 0 \text{ V}$ , $T_A = 0^\circ \text{ to } +70^\circ \text{C}$ )

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	$V_{DD}$	-0.3 to +7.0	V
INPUT VOLTAGE	$V_{IN}$	-0.3 to +7.0	V
OPERATING TEMP.	$T_A$	0 to +70	°C
STORAGE TEMP.	$T_{STG}$	-55 to +150	°C

### ■ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
$\overline{IRQ}$ *	Interrupt Request
RDY *	Ready
ML	Memory Lock
$\overline{NMI}$ *	Non-Maskable Interrupt
SYNC	Synchronize
$\overline{RES}$ *	Reset
$\overline{SO}$ *	Set Overflow
NC	No Connection
R/W	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
$\phi_0$	Clock Input
$\phi_1, \phi_2$	Clock Output

\*This pin has an optional internal pullup for a No Connect condition.

### ■ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage $\phi_0$ (IN)	$V_{IH}$	$V_{SS} + 2.4$	—	$V_{DD}$	V
Input High Voltage $\overline{RES}$ , $\overline{NMI}$ , RDY, $\overline{IRQ}$ , Data, S.O.		$V_{SS} + 2.0$	—	—	V
Input Low Voltage $\phi_0$ (IN)	$V_{IL}$	$V_{SS} - 0.3$	—	$V_{SS} + 0.4$	V
$\overline{RES}$ , $\overline{NMI}$ , RDY, $\overline{IRQ}$ , Data, S.O.		—	—	$V_{SS} + 0.8$	V
Input Leakage Current ( $V_{IN} = 0$ to 5.25V, $V_{DD} = 5.25\text{V}$ )	$I_{IN}$	—	—	—	—
With pullups		-30	—	+30	$\mu\text{A}$
Without pullups		—	—	+1.0	$\mu\text{A}$
Three State (Off State) Input Current ( $V_{IN} = 0.4$ to 2.4V, $V_{CC} = 5.25\text{V}$ )		—	—	—	—
Data Lines	$I_{TSI}$	—	—	10	$\mu\text{A}$
Output High Voltage ( $I_{OH} = -100 \mu\text{A}$ dc, $V_{DD} = 4.75\text{V}$ )		—	—	—	—
SYNC, Data, A0-A15, R/W)	$V_{OH}$	$V_{SS} + 2.4$	—	—	V
Out Low Voltage ( $I_{OL} = 1.6\text{mA}$ dc, $V_{DD} = 4.75\text{V}$ )		—	—	—	—
SYNC, Data, A0-A15, R/W)	$V_{OL}$	—	—	$V_{SS} + 0.4$	V
Supply Current $f = 1\text{MHz}$	$I_{DD}$	—	—	4	mA
Supply Current $f = 2\text{MHz}$	$I_{DD}$	—	—	8	mA
Capacitance ( $V_{IN} = 0$ , $T_A = 25^\circ\text{C}$ , $f = 1\text{MHz}$ )	C	—	—	—	pF
Logic	$C_{IN}$	—	—	5	
Data		—	—	10	
A0-A15, R/W, SYNC	$C_{out}$	—	—	10	
$\phi_0$ (IN)	$C_{\phi_0}$ (IN)	—	—	10	

# ■ **AC CHARACTERISTICS** $V_{DD} = 5.0V \pm 5\%$ , $T_A = 0^\circ C$ to $70^\circ C$ , Load = 1 TTL + 130 pF

Parameter	Symbol	1MHZ		2MHZ		3MHZ		Unit
		Min	Max	Min	Max	Min	Max	
Delay Time, $\theta_0$ (IN) to $\theta_2$ (OUT)	$t_{DLY}$	—	60	—	60	20	60	nS
Delay Time, $\theta_1$ (OUT) to $\theta_2$ (OUT)	$t_{DLY1}$	—20	20	—20	20	—20	20	nS
Cycle Time	$t_{CYC}$	1.0	5000*	0.50	5000*	0.33	5000*	$\mu S$
Clock Pulse Width Low	$t_{PL}$	460	—	220	—	160	—	nS
Clock Pulse Width High	$t_{PH}$	460	—	220	—	160	—	nS
Fall Time, Rise Time	$t_F, t_R$	—	25	—	25	—	25	nS
Address Hold Time	$t_{AH}$	20	—	20	—	0	—	nS
Address Setup Time	$t_{ADS}$	—	225	—	140	—	110	nS
Access Time	$t_{ACC}$	650	—	310	—	170	—	nS
Read Data Hold Time	$t_{DHR}$	10	—	10	—	10	—	nS
Read Data Setup Time	$t_{DSU}$	100	—	60	—	60	—	nS
Write Data Delay Time	$t_{MDS}$	—	30	—	30	—	30	nS
Write Data Hold Time	$t_{DHW}$	20	—	20	—	15	—	nS
SO Setup Time	$t_{SO}$	100	—	100	—	100	—	nS
Processor Control Setup Time**	$t_{PCS}$	200	—	150	—	150	—	nS
SYNC Setup Time	$t_{SYNC}$	—	225	—	140	—	100	nS
ML Setup Time	$t_{ML}$	—	225	—	140	—	100	nS
Input Clock Rise/Fall Time	$t_{F\theta_0}, t_{R\theta_0}$	—	25	—	25	—	25	nS

\*NCR65C02 can be held static with  $\theta_2$  high.

\*\*This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

## ■ **MICROPROCESSOR OPERATIONAL ENHANCEMENTS**

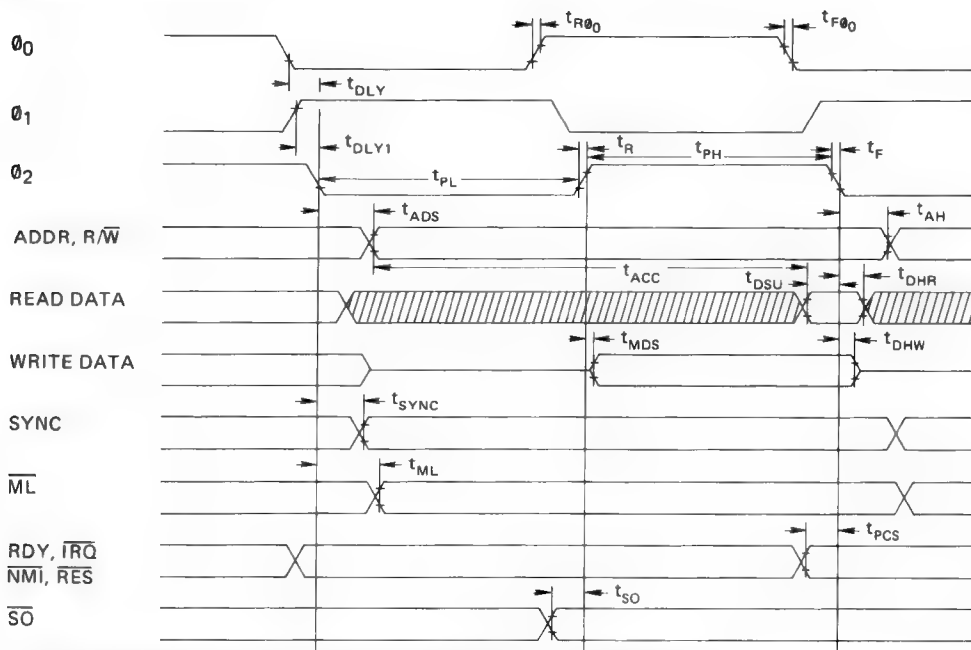
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor																					
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.																					
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). <table> <tr> <th>Op Code</th><th>Bytes</th><th>Cycles</th></tr> <tr> <td>X2</td><td>2</td><td>2</td></tr> <tr> <td>X3, X7, XB, XF</td><td>1</td><td>1</td></tr> <tr> <td>44</td><td>2</td><td>3</td></tr> <tr> <td>54, D4, F4</td><td>2</td><td>4</td></tr> <tr> <td>5C</td><td>3</td><td>8</td></tr> <tr> <td>DC, FC</td><td>3</td><td>4</td></tr> </table>	Op Code	Bytes	Cycles	X2	2	2	X3, X7, XB, XF	1	1	44	2	3	54, D4, F4	2	4	5C	3	8	DC, FC	3	4
Op Code	Bytes	Cycles																					
X2	2	2																					
X3, X7, XB, XF	1	1																					
44	2	3																					
54, D4, F4	2	4																					
5C	3	8																					
DC, FC	3	4																					
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.																					
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.																					
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.																					
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.																					
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.																					

## ■ **MICROPROCESSOR HARDWARE ENHANCEMENTS**

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during $\theta_2$ .
Unused input-only pins ( $\overline{IRQ}$ , NMI, RDY, RES, SO).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to $V_{DD}$ (approximately 250 K ohm.)

## NCR65C02

### ■ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

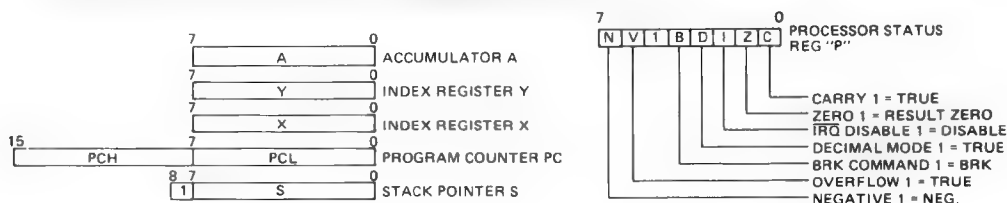
### ■ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG, X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

### ■ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND, X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

## ■ MICROPROCESSOR PROGRAMMING MODEL



## ■ FUNCTIONAL DESCRIPTION

### Timing Control

The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

### Program Counter

The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

### Instruction Register and Decode

Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

### Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

### Accumulator

The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

### Index Registers

There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

### Stack Pointer

The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts (NMI and IRQ). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

### Processor Status Register

The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

## NCR65C02

### ■ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

#### **Implied Addressing [Implied]**

In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

#### **Accumulator Addressing [Accum]**

This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

#### **Immediate Addressing [Immediate]**

With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

#### **Absolute Addressing [Absolute]**

For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

#### **Zero Page Addressing [Zero Page]**

Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

#### **Absolute Indexed Addressing [ABS, X or ABS, Y]**

Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

#### **Zero Page Indexed Addressing [ZPG, X or ZPG, Y]**

Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

#### **Relative Addressing [Relative]**

Relative addressing is used only with branch instructions;

it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is -128 to +127 bytes from the next instruction.

#### **Zero Page Indexed Indirect Addressing [(IND, X)]**

With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

#### **\*Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**

With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

#### **Indirect Indexed Addressing [(IND), Y]**

This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

#### **\*Zero Page Indirect Addressing [(ZPG)]**

In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

#### **Absolute Indirect Addressing [(ABS)]**

##### **(Jump Instruction Only)**

The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: \* = New Address Modes

## ■ SIGNAL DESCRIPTION

### Address Bus (A0-A15)

A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

### Clocks ( $\theta_0$ , $\theta_1$ , and $\theta_2$ )

$\theta_0$  is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The  $\theta_2$  clock output is in phase with  $\theta_0$ . The  $\theta_1$  output pin is 180° out of phase with  $\theta_0$ . (See timing diagram.)

### Data Bus (D0-D7)

The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

### Interrupt Request ( $\overline{\text{IRQ}}$ )

This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The  $\overline{\text{IRQ}}$  is sampled during  $\theta_2$  operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during  $\theta_1$ . The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further  $\overline{\text{IRQ}}$ s may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

### Memory Lock ( $\overline{\text{ML}}$ )

In a multiprocessor system, the  $\overline{\text{ML}}$  output indicates the need to defer the arbitration of the next bus cycle to ensure the integrity of read-modify-write instructions.  $\overline{\text{ML}}$  goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

### Non-Maskable Interrupt ( $\overline{\text{NMI}}$ )

A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The  $\overline{\text{NMI}}$  is sampled during  $\theta_2$ ; the current instruction is completed and the interrupt sequence begins during  $\theta_1$ . The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

**Note:** Since this interrupt is non-maskable, another  $\overline{\text{NMI}}$  can occur before the first is finished. Care should be taken when using  $\overline{\text{NMI}}$  to avoid this.

### Ready (RDY)

This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one ( $\theta_1$ ), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two ( $\theta_2$ ) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

### Reset ( $\overline{\text{RES}}$ )

This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after VDD reaches operating voltage from a power down. A positive transition on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on  $\overline{\text{RES}}$ .

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

### Read/Write ( $\text{R}/\overline{\text{W}}$ )

This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

### Set Overflow ( $\overline{\text{SO}}$ )

A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of  $\theta_1$ .

### Synchronize (SYNC)

This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during  $\theta_1$  of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the  $\theta_1$  clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

## NCR65C02

### ■ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC	Add Memory to Accumulator with Carry	LDX	Load Index X with Memory
AND	"AND" Memory with Accumulator	LDY	Load Index Y with Memory
ASL	Shift One Bit Left	LSR	Shift One Bit Right
BCC	Branch on Carry Clear	NOP	No Operation
BCS	Branch on Carry Set	ORA	"OR" Memory with Accumulator
BEQ	Branch on Result Zero	PHA	Push Accumulator on Stack
BIT	Test Memory Bits with Accumulator	PHP	Push Processor Status on Stack
BMI	Branch on Result Minus	*PHX	Push Index X on Stack
BNE	Branch on Result not Zero	*PHY	Push Index Y on Stack
BPL	Branch on Result Plus	PLA	Pull Accumulator from Stack
*BRA	Branch Always	PLP	Pull Processor Status from Stack
BRK	Force Break	*PLX	Pull Index X from Stack
BVC	Branch on Overflow Clear	*PLY	Pull Index Y from Stack
BVS	Branch on Overflow Set	ROL	Rotate One Bit Left
CLC	Clear Carry Flag	ROR	Rotate One Bit Right
CLD	Clear Decimal Mode	RTI	Return from Interrupt
CLI	Clear Interrupt Disable Bit	RTS	Return from Subroutine
CLV	Clear Overflow Flag	SBC	Subtract Memory from Accumulator with Borrow
CMP	Compare Memory and Accumulator	SEC	Set Carry Flag
CPX	Compare Memory and Index X	SED	Set Decimal Mode
CPY	Compare Memory and Index Y	SEI	Set Interrupt Disable Bit
*DEA	Decrement Accumulator	STA	Store Accumulator in Memory
DEC	Decrement by One	STX	Store Index X in Memory
DEX	Decrement Index X by One	STY	Store Index Y in Memory
DEY	Decrement Index Y by One	*STZ	Store Zero in Memory
EOR	"Exclusive-or" Memory with Accumulator	TAX	Transfer Accumulator to Index X
*INA	Increment Accumulator	TAY	Transfer Accumulator to Index Y
INC	Increment by One	*TRB	Test and Reset Memory Bits with Accumulator
INX	Increment Index X by One	*TSB	Test and Set Memory Bits with Accumulator
INY	Increment Index Y by One	TSX	Transfer Stack Pointer to Index X
JMP	Jump to New Location	TXA	Transfer Index X to Accumulator
JSR	Jump to New Location Saving Return Address	TXS	Transfer Index X to Stack Pointer
LDA	Load Accumulator with Memory	TYA	Transfer Index Y to Accumulator

Note: \* = New Instruction

### ■ MICROPROCESSOR OP CODE TABLE

S	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK	ORA ind, X				TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs	0
1	BPL rel	ORA ind, Y	ORA*† (zpg)			TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X	1
2	JSR abs	AND ind, X				BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs	2
3	BMI rel	AND ind, Y	AND*† (zpg)			BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X	3
4	RTI	EOR ind, X					EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs	4
5	BVC rel	EOR ind, Y	EOR*† (zpg)				EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY* A			EOR abs, X	LSR abs, X	5
6	RTS	ADC ind, X				STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs	6
7	BVS rel	ADC ind, Y	ADC*† (zpg)			STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY* abs (ind, X)		JMP*† abs (ind, X)	ADC abs, X	ROR abs, X	7
8	BRA* rel	STA ind, X				STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA A		STY abs	STA abs	STX abs	8
9	BCC rel	STA ind, Y	STA*† (zpg)			STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS A		STZ* abs	STA abs, X	STZ* abs, X	9
A	LDY imm	LDA ind, X	LDX imm			LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX A		LDY abs	LDA abs	LDX abs	A
B	BCS rel	LDA ind, Y	LDA*† (zpg)			LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX A		LDY abs, X	LDA abs, X	LDX abs, Y	B
C	CPY imm	CMP ind, X				CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX A		CPY abs	CMP abs	DEC abs	C
D	BNE rel	CMP ind, Y	CMP*† (zpg)				CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX* A			CMP abs, X	DEC abs, X	D
E	CPX imm	SBC ind, X				CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP A		CPX abs	SBC abs	INC abs	E
F	BEQ rel	SBC ind, Y	SBC*† (zpg)				SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX* A			SBC abs, X	INC abs, X	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: \* = New OP Codes

Note: † = New Address Modes

## ■ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

[illegible]

## Notes

1. Add 1 to "n" if page boundary is crossed.
2. Add 1 to "n" if branch occurs to same page.  
Add 2 to "n" if branch occurs to different page.
3. Add 1 to "n" if decimal mode.
4. V bit equals memory bit 6 prior to execution.  
N bit equals memory bit 7 prior to execution.

X Index X

Y Index Y

**A Accumulator**

**M** Memory per effective address

Ms Memory per stack pointer

+ Add

- Subtract

^ And

**V Or**

✚ Exclusive or

n No. Cycles

# No. Bytes

**M6** Memory bit 6

**M7** Memory bit 7

- \*5. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.

---

---

## Appendix B

---

---

# Memory Map

This appendix lists all important RAM and hardware locations in address order and briefly describes them. Appendix C contains a similar list for important firmware addresses.

The tables in this appendix list addresses in either two or three forms: the hexadecimal form (preceded by a dollar sign) for use in assembly language; the decimal form for use in Applesoft BASIC; and (for numbers greater than 32,767) the complementary decimal value for use in Apple Integer BASIC.

---

---

### Page \$00

Table B-1 lists the zero page addresses in hexadecimal and decimal form, followed by symbols denoting the firmware or system software that uses them.

- ☐ *M* denotes the monitor.
- ☐ *A* denotes Applesoft BASIC.
- ☐ *I* denotes Integer BASIC.
- ☐ *D* denotes DOS 3.3.
- ☐ *P* denotes ProDOS. Locations whose contents ProDOS saves and restores afterward have a P in parentheses, indicating that ProDOS has no net effect on them.

**Table B-1**  
Page \$00 use

Hex	Dec	Used by		Hex	Dec	Used by	
\$00	0	A		\$30	48	M	
\$01	1	A		\$31	49	M	
\$02	2	A		\$32	50	M	
\$03	3	A		\$33	51	M	
\$04	4	A		\$34	52	M	
\$05	5	A		\$35	53	M	D
\$06	6			\$36	54	M	D
\$07	7			\$37	55	M	D
\$08	8			\$38	56	M	D
\$09	9			\$39	57	M	D
\$0A	10	A		\$3A	58	M	P
\$0B	11	A		\$3B	59	M	P
\$0C	12	A		\$3C	60	M	P
\$0D	13	A		\$3D	61	M	P
\$0E	14	A		\$3E	62	M	D P
\$0F	15	A		\$3F	63	M	D P
\$10	16	A		\$40	64	M	D (P)
\$11	17	A		\$41	65	M	D (P)
\$12	18	A		\$42	66	M	D (P)
\$13	19	A		\$43	67	M	D (P)
\$14	20	A		\$44	68	M	D (P)
\$15	21	A		\$45	69	M	D (P)
\$16	22	A		\$46	70	M	D (P)
\$17	23	A		\$47	71	M	D (P)
\$18	24	A		\$48	72	M	D (P)
\$19	25			\$49	73	M	(P)
\$1A	26			\$4A	74	I	D (P)
\$1B	27			\$4B	75	I	D (P)
\$1C	28			\$4C	76	I	D (P)
\$1D	29			\$4D	77	I	D (P)
\$1F	31			\$4F	79	M	
\$25	37	M		\$55	85	M A I	
\$26	38	M	D	\$56	86	A I	
\$27	39	M	D	\$57	87	A I	
\$28	40	M	D	\$58	88	A I	
\$29	41	M	D	\$59	89	A I	
\$2A	42	M	D	\$5A	90	A I	
\$2B	43	M	D	\$5B	91	A I	
\$2C	44	M	D	\$5C	92	A I	
\$2D	45	M	D	\$5D	93	A I	
\$2E	46	M	D	\$5E	94	A I	
\$2F	47	M	D	\$5F	95	A I	

**Table B-1 (continued)**  
Page \$00 use

Hex	Dec	Used by	Hex	Dec	Used by
\$60	96	A I	\$90	144	A I
\$61	97	A I	\$91	145	A I
\$62	98	A I	\$92	146	A I
\$63	99	A I	\$93	147	A I
\$64	100	A I	\$94	148	A I
\$65	101	A I	\$95	149	A I
\$66	102	A I	\$96	150	A I
\$67	103	A I D	\$97	151	A I
\$68	104	A I D	\$98	152	A I
\$69	105	A I D	\$99	153	A I
\$6A	106	A I D	\$9A	154	A I
\$6B	107	A I	\$9B	155	A I
\$6C	108	A I	\$9C	156	A I
\$6D	109	A I	\$9D	157	A I
\$6E	110	A I	\$9E	158	A I
\$6F	111	A I D	\$9F	159	A I
\$70	112	A I D	\$A0	160	A I
\$71	113	A I	\$A1	161	A I
\$72	114	A I	\$A2	162	A I
\$73	115	A I	\$A3	163	A I
\$74	116	A I	\$A4	164	A I
\$75	117	A I	\$A5	165	A I
\$76	118	A I	\$A6	166	A I
\$77	119	A I	\$A7	167	A I
\$78	120	A I	\$A8	168	A I
\$79	121	A I	\$A9	169	A I
\$7A	122	A I	\$AA	170	A I
\$7B	123	A I	\$AB	171	A I
\$7C	124	A I	\$AC	172	A I
\$7D	125	A I	\$AD	173	A I
\$7E	126	A I	\$AE	174	A I
\$7F	127	A I	\$AF	175	A I D
\$80	128	A I	\$B0	176	A I D
\$81	129	A I	\$B1	177	A I
\$82	130	A I	\$B2	178	A I
\$83	131	A I	\$B3	179	A I
\$84	132	A I	\$B4	180	A I
\$85	133	A I	\$B5	181	A I
\$86	134	A I	\$B6	182	A I
\$87	135	A I	\$B7	183	A I
\$88	136	A I	\$B8	184	A I
\$89	137	A I	\$B9	185	A I

**Table B-1 (continued)**  
Page \$00 use

Hex	Dec	Used by	Hex	Dec	Used by
\$8A	138	A I	\$BA	186	A I
\$8B	139	A I	\$BB	187	A I
\$8C	140	A I	\$BC	188	A I
\$8D	141	A I	\$BD	189	A I
\$8E	142	A I	\$BE	190	A I
\$8F	143	A I	\$BF	191	A I
\$C0	192	A I	\$E0	224	A
\$C1	193	A I	\$E1	225	A
\$C2	194	A I	\$E2	226	A
\$C3	195	A I	\$E3	227	
\$C4	196	A I	\$E4	228	A
\$C5	197	A I	\$E5	229	A
\$C6	198	A I	\$E6	230	A
\$C7	199	A I	\$E7	231	A
\$C8	200	A I	\$E8	232	A
\$C9	201	A I	\$E9	233	A
\$CA	202	A I D	\$EA	234	A
\$CB	203	A I D	\$EB	235	
\$CC	204	A I D	\$EC	236	
\$CD	205	A I D	\$ED	237	
\$CE	206	I	\$EE	238	
\$CF	207	I	\$EF	239	
\$D0	208	A I	\$F0	240	A
\$D1	209	A I	\$F1	241	A
\$D2	210	A I	\$F2	242	A
\$D3	211	A I	\$F3	243	A
\$D4	212	A I	\$F4	244	A
\$D5	213	A I	\$F5	245	A
\$D6	214	I	\$F6	246	A
\$D7	215	I	\$F7	247	A
\$D8	216	A I D	\$F8	248	A
\$D9	217	A I	\$F9	249	
\$DA	218	A I	\$FA	250	
\$DB	219	A I	\$FB	251	
\$DC	220	A I	\$FC	252	
\$DD	221	A I	\$FD	253	
\$DE	222	A I	\$FE	254	
\$DF	223	A I	\$FF	255	

---

---

## Page \$03

Most of page \$03 is available for small machine-language programs. The built-in Monitor uses the top 16 addresses of page \$03, as shown in Figure B-2; the XFer routine uses locations \$03ED and \$03EE. If you are using DOS or ProDOS, it also uses the 32 locations \$03D0 through \$03EF.

**Table B-2**  
Page \$03 use

Hex	Dec	Use
\$03F0	1008	Address of BRK request handler (normally \$59, \$FA)
\$03F1	1009	
\$03F2	1010	Reset vector
\$03F3	1011	Power-up byte (see text)
\$03F4	1012	
\$03F5	1013	
\$03F6	1014	
\$03F7	1015	Jump instruction to Applesoft &-command handler (initially \$4C, \$58, \$FF)
\$03F8	1016	
\$03F9	1017	
\$03FA	1018	
\$03FB	1019	Jump instruction to user Control-Y command handler
\$03FC	1020	
\$03FD	1021	
\$03FE	1022	Address of user IRQ interrupt handler
\$03FF	1023	

---

---

## Screen holes

One result of the way the Apple IIc hardware maps display memory on the screen is that groups of 8 memory addresses are left over in 16 areas of the text and low-resolution display pages—8 areas in main RAM and 8 in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4.

## Memory expansion

The version of the Apple IIc that supports the memory expansion card uses some of the screen holes differently than earlier versions. Where they differ, the memory expansion ROM assignments are given in parentheses in Tables B-3 and B-4 following the original and UniDisk 3.5 assignments.

**Table B-3**

Main memory screen hole allocations

Hex	Dec	Description
\$0478	1144	Mouse port: low byte of clamping minimum
\$0479	1145	Reserved for serial port 1
\$047A	1146	Reserved for serial port 2
\$047B	1147	Reserved
\$047C	1148	Low byte of X coordinate (Reserved)
\$047D	1149	Reserved for mouse port
\$047E	1150	Reserved
\$047F	1151	Reserved (Low byte of X coordinate)
\$04F8	1272	Mouse port: low byte of clamping maximum
\$04F9	1273	Reserved for serial port 1
\$04FA	1274	Reserved for serial port 2
\$04FB	1275	Reserved
\$04FC	1276	Low byte of Y coordinate (Reserved)
\$04FD	1277	Reserved for mouse port
\$04FE	1278	Reserved
\$04FF	1279	Reserved (Low byte of Y coordinate)
\$0578	1400	Mouse port: high byte of clamping minimum
\$0579	1401	Port 1 printer width (1-255; 0 = unlimited)
\$057A	1402	Port 2 line length (1-255; 0 = unlimited)
\$057B	1403	Cursor horizontal position (80-column display)
\$057C	1404	High byte of X coordinate (Reserved)
\$057D	1405	Reserved for mouse port
\$057E	1406	Reserved
\$057F	1407	Reserved (High byte of X coordinate)
\$05F8	1528	Mouse port: high byte of clamping maximum
\$05F9	1529	Port 1 temporary storage location
\$05FA	1530	Port 2 temporary storage location
\$05FB	1531	Reserved
\$05FC	1532	High byte of Y coordinate (Reserved)
\$05FD	1533	Reserved for mouse port
\$05FE	1534	Reserved
\$05FF	1535	Reserved (High byte of Y coordinate)

**Table B-3 (continued)**  
Main memory screen hole allocations

Hex	Dec	Description
\$0678	1656	Reserved
\$0679	1657	Indicates when port 1 firmware is parsing a command
\$067A	1658	Indicates when port 2 firmware is parsing a command
\$067B	1659	Reserved
\$067C	1660	Mouse port: reserved (Reserved)
\$067D	1661	Reserved for mouse port
\$067E	1662	Reserved
\$067F	1663	Reserved (Mouse port: reserved)
\$06F8	1784	Reserved
\$06F9	1785	Current port 1 command character
\$06FA	1786	Current port 2 command character
\$06FB	1787	Reserved
\$06FC	1788	Mouse port: reserved (Reserved)
\$06FD	1789	Reserved for mouse port
\$06FE	1790	Reserved
\$06FF	1791	Reserved (Mouse port: reserved)
\$0778	1912	DEVNO: \$n0 = current active port number x 16
\$0779	1913	Port 1 flags for echo and auto line feed
\$077A	1914	Port 2 flags for each and auto line feed
\$077B	1915	Reserved
\$077C	1916	Mouse port status byte (Reserved)
\$077D	1917	Reserved for mouse port
\$077E	1918	Reserved
\$077F	1919	Reserved (Mouse port status byte)
\$07F8	2040	MSLOT: owner of \$C800–\$CFFF (\$C3, video)
\$07F9	2041	Port 1 current printer column
\$07FA	2042	Port 2 current line position
\$07FB	2043	Reserved
\$07FC	2044	Mouse port mode byte (Reserved)
\$07FD	2045	Reserved for mouse port
\$07FE	2046	Reserved
\$07FF	2047	Reserved (Mouse port mode byte)

**Table B-4**  
Auxiliary memory screen hole allocations

Hex	Dec	Description
\$0478	1144	Initial port 1 ACIA control register values (\$9E)
\$0479	1145	Initial port 1 ACIA command register values (\$0B)
\$047A	1146	Initial port 1 characteristics flags (\$40)
\$047B	1147	Initial port 1 printer width (\$50)
\$047C	1148	Initial port 2 ACIA control register values (\$16)
\$047D	1149	Initial port 2 ACIA command register values (\$0B)
\$047E	1150	Initial port 2 characteristics flags (\$01)
\$047F	1151	Initial port 2 line length (\$00)
\$04F8 through \$04FF	1272  1279	Reserved
\$0578 through \$057F	1400  1407	Reserved
\$05F8 through \$05FF	1528  1535	Reserved
\$0678 through \$067F	1656  1663	Reserved
\$06F8 through \$06FF	1784  1791	Reserved
\$0778 through \$077F	1912  1919	Reserved
\$07F8 through \$07FF	2040  2047	Reserved

## The hardware page

Tables B-5 through B-9 list all the hardware locations available for use in the Apple IIc. These tables have a column at the left that is not present in other tables. This column, labeled *RW*, indicates the action to take at a particular location.

- ☐ *R* means read.
- ☐ *RR* means read twice in succession.
- ☐ *R7* means read the byte and then check bit 7; in the use column, "See if..." refers to the condition represented by bit 7 = 1, unless otherwise specified. Bit 7 has a value of \$80, so if the contents of the location are greater than or equal to \$80, the bit is on.

Another way to test bit 7 (the sign bit) is with a BIT instruction, followed by BPL (bit 7 was 0) or BMI (bit 7 was 1).

- ☐ *R/W* means to either read or write. For writing, the value is unimportant.
- ☐ *W* means to write only. The value is unimportant.
- ☐ *N* means not to read or write, because the location is reserved.

An address of the form \$C00x refers to the 16 locations from \$C000 through \$C00F. Labels, when they are shown, are simply memory aids. Some of them correspond to the labels at those addresses in the firmware, others do not. Your program will have to assign a label for it anyway.

**Table B-5**  
Addresses \$C000–\$C03F

<b>RW</b>	<b>Hex</b>	<b>Dec</b>	<b>Neg dec</b>	<b>Label</b>	<b>Use</b>
R	\$C00x			KStrb	Read keyboard data (bits 0-6) and strobe (bit 7)
W	\$C000	49152	-16384	80Store	Off: Page2 switches Page 1 and 2
W	\$C001	49153	-16383	80Store	On: Page2 switches Page 1 and 1X
W	\$C002	49154	-16382	RAMRd	Off: Read main 48K RAM
W	\$C004	49156	-16380	RAMWrt	Off: Write in main 48K RAM
W	\$C005	49157	-16379	RAMWrt	On: Write in auxiliary 48K RAM
W	\$C006	49158	-16378		Reserved
W	\$C007	49159	-16377		Reserved
W	\$C008	49160	-16376	AltZP	Off: Use main P0, P1, bank-switched RAM
W	\$C009	49161	-16375	AltZP	On: Use auxiliary P0, P1, bank-switched RAM
W	\$C00A	49162	-16374		Reserved
W	\$C00B	49163	-16373		Reserved
W	\$C00C	49164	-16372	80Col	Off: 40-column display

Table B-5 (continued)  
Addresses \$C000-\$C03F

RW	Hex	Dec	Neg dec	Label	Use
W	\$C00D	49165	-16371	80Col	On: 80-column display
W	\$C00E	49166	-16270	AltChar	Off: Display primary character set
W	\$C00F	49167	-16369	AltChar	On: Display alternate character set
W	\$C01x				Clear keyboard strobe (\$C00x bit 7)
R7	\$C010	49168	-16368	AKD	See if any key now down; clear strobe
R7	\$C011	49169	-16367	RdBnk2	See if using \$D000 bank 2 (or 1)
R7	\$C012	49170	-16366	RdLCRAM	See if reading RAM (or ROM).
R7	\$C013	49171	-16365	RdRAMRd	See if reading auxiliary 48K RAM (or main)
R7	\$C014	49172	-16364	RdRAMWrt	See if writing auxiliary 48K RAM (or main)
R	\$C015	49173	-16363	RstXInt	Reset mouse X0 interrupt
R7	\$C016	49174	-16362	RdAltZP	See if auxiliary P0, P1 and bank-switched RAM
R	\$C017	49175	-16361	RstYInt	Reset mouse Y interrupt
R7	\$C018	49176	-16360	Rd80Store	See if 80Store on (or off)
R7	\$C019	49177	-16359	RstVBI	See if VBIInt off (1); reset it
R7	\$C01A	49178	-16358	RdTEXT	See if text (or graphics)
R7	\$C01B	49179	-16357	RdMIXED	See if mixed mode switch on
R7	\$C01C	49180	-16356	RdPage2	See if Page 2/1X selected (or 1)
R7	\$C01D	49181	-16355	RdHiRes	See if high-resolution switch on
R7	\$C01E	49182	-16354	RdAltChar	See if alternate character set (or primary)
R7	\$C01F	49183	-16353	Rd80Col	See if 80-column hardware on
N	\$C020	49184	-16352		
	through				Reserved (read and write)
N	\$C02F	49199	-16337		
W	\$C030	49200	-16336	Reserved	
R	\$C030	49200	-16336		Toggle speaker
N	\$C031	49201	-16335		
	through				Reserved (read and write)
N	\$C03F	49215	-16321		

**Table B-6**  
Addresses \$C040–\$C05F

RW	Hex	Dec	Neg dec	Label	Use
R7	\$C040	49216	–16320	RdXYMsk	See if X0/Y0 mask set
R7	\$C041	49217	–16319	RdVBLMsk	See if VBL mask set
R7	\$C042	49218	–16318	RdX0Edge	See if interrupt on falling X0 edge
R7	\$C043	49219	–16317	RdY0Edge	See if interrupt on falling Y0 edge
N	\$C044	49220	–16316		Reserved
N	\$C045	49221	–16315)		Reserved
N	\$C046	49222	–16314		Reserved
N	\$C047	49223	–16313		Reserved
R	\$C048	49224	–16312	RstXY	Reset X0/Y0 interrupt flags
N	\$C049	49225	–16311		Reserved
N	\$C04A	49226	–16310		Reserved
N	\$C04B	49227	–16309		Reserved
N	\$C04C	49228	–16308		Reserved
N	\$C04D	49229	–16307		Reserved
N	\$C04E	49230	–16306		Reserved
N	\$C04F	49231	–16305		Reserved
R/W	\$C050	49232	–16304	TEXT	Off: Graphics display
R/W	\$C051	49233	–16303	TEXT	On: Text display
R/W	\$C052	49234	–16302	MIXED	Off: Text or graphics only
R/W	\$C053	49235	–16301	MIXED	On: Combination text and graphics
R/W	\$C054	49236	–16300	Page2	Off: Use Page 1
R/W	\$C055	49237	–16299	Page2	On: Display Page 2 (80Store off); store to Page 1X (80Store on)
R/W	\$C056	49238	–16298	HiRes	Off: Low resolution
R/W	\$C057	49239	–16297	HiRes	On: High resolution; double if 80Col and DHiRes on
N	\$C058	49240	–16296		Reserved if IOUDis on (\$C07E bit 7=1)
R/W				DisX	Disable (mask) mouse X0/Y0 interrupts
N	\$C059	49241	–16295		Reserved if IOUDis on
R/W				EnbXY	Enable (allow) mouse X0/Y0 interrupts
N	\$C05A	49242	–16294		Reserved if IOUDis on
R/W				DisVBL	Disable (mask) VBL interrupts
N	\$C05B	49243	–16293		Reserved if IOUDis on
R/W				EnVBL	Enable (allow) VBL interrupts
N	\$C05C	49244	–16292		Reserved if IOUDis on
R/W				X0Edge	Interrupt on rising edge of X0
N	\$C05D	49245	–1629		Reserved if IOUDis on
R/W				X0Edge	Interrupt on falling edge of X0
R/W	\$C05E	49246	–16290	DHiRes	If IOUDis on: Set double high-resolution
R/W				Y0Edge	If IOUDis off: Interrupt on rising Y0
R/W	\$C05F	49247	–16289	DHiRes	If IOUDis on: Clear double high-resolution
R/W				Y0Edge	If IOUDis off: Interrupt on falling Y0

**Table B-7**  
Addresses \$C060-\$C07F

RW	Hex	Dec	Neg dec	Label	Use
W	\$C06x				Reserved (write)
R7	\$C060	4924	-16288	Rd80Sw	See if 80/40 switch down (= 40)
R7	\$C061	49249	-16287	RdBtn0	See if mouse button/Open-Apple pressed
R7	\$C062	49250	-16286	RdBtn1	See if switch 1/Solid Apple pressed
R7	\$C063	49251	-16285	Rd63	See if mouse button not pressed
R7	\$C064	49252	-16284	Pdl0	See if hand control button 0 pressed
R7	\$C065	49253	-16283	Pdl1	See if hand control button 1 pressed
R7	\$C066	49254	-16282	MouX1	See if mouse X1 (direction) is high
R7	\$C067	49255	-16281	MouY1	See if mouse Y1 (direction) is high
N	\$C068	49256	-16280		
	through				Reserved (write and read)
N	\$C06F	49263	-16273		
R/W	\$C07x				Trigger paddle timer; reset VBlInt; however, some \$C07x are reserved
R/W	\$C070	49264	-16272	PTrig	Designated trigger or reset location
N	\$C071	49265	-16271		
	through				Reserved
N	\$C07D	49277	-16259		
R7	\$C07E	49278	-16258	RdIOUDis	See if IOUDis on; trigger paddle timer; reset VBlInt
W				IOUDis	On: Enable access to DHiRes switch; disable \$C058-\$C05F IOU access
R7	\$C07F	49279	-16257	RdDHiRes	See if DHiRes on
W				IOUDis	Off: Disable access to DHiRes switch; enable \$C058-\$C05F IOU access

**Table B-8**  
Addresses \$C080-\$C0AF

RW	Hex	Dec	Neg dec	Label	Use
R	\$C080	49280	-16256		Read RAM; no write; use \$D000 bank 2
RR	\$C081	49281	-16255		Read ROM; write RAM; use \$D000 bank 2
R	\$C082	49282	-16254		Read ROM; no write; use \$D000 bank 2
RR	\$C083	49283	-16253		Read and write RAM; use \$D000 bank 2
N	\$C084	49284	-16252		Reserved
N	\$C085	49285	-16251		Reserved
N	\$C086	49286	-16250		Reserved
N	\$C087	49287	-16249		Reserved
R	\$C088	49288	-16248		Read RAM; no write; use \$D000 bank 1
RR	\$C089	49289	-16247		Read ROM; write RAM; use \$D000 bank 1
R	\$C08A	49290	-16246		Read ROM; no write; use \$D000 bank 1
RR	\$C08B	49291	-16245		Read and write RAM; use \$D000 bank 1
N	\$C08C	49292	-16244		Reserved
N	\$C08D	49293	-16243		Reserved
N	\$C08E	49294	-16242		Reserved
N	\$C08F	49295	-16241		Reserved
N	\$C090	49296	-16240		Reserved
	through				
N	\$C097	49303	-16233		
R/W	\$C098	49304	-16232		Port 1 ACIA transmit/receive register
R/W	\$C099	49305	-16231		Port 1 ACIA status register
R/W	\$C09A	49306	-16230		Port 1 ACIA command register
R/W	\$C09B	49307	-16229		Port 1 ACIA control register
N	\$C09C	49308	-16228		Reserved
	through				
N	\$C09F	49311	-16225		
N	\$C0A0	49312	-16224		Reserved
	through				
N	\$C0A7	49319	-16217		
R/W	\$C0A8	49320	-16216		Port 2 ACIA transmit/receive register
R/W	\$C0A9	49321	-16215		Port 2 ACIA status register
R/W	\$C0AA	49322	-16214		Port 2 ACIA command register
R/W	\$C0AB	49323	-16213		Port 2 ACIA control register
N	\$C0AC	49324	-16212		Reserved
	through				
N	\$C0AF	49327	-16209		

**Table B-9**  
Addresses \$C0B0-\$C0FF

RW	Hex	Dec	Neg Dec	Label	Use
N	\$C0B0 through	49328	-16208		Reserved
N	\$C0BF	49343	-16193		
N	\$C0C0 through	49344	-16192		Reserved
N	\$C0CF	49359	-16177		
N	\$C0D0 through	49360	-16176		Reserved
N	\$C0DF	49375	-16161		
N	\$C0E0 through	49376	-16160		Reserved
N	\$C0EF	49391	-16145		
N	\$C0F0 through	49392	-16144		Reserved
N	\$C0FF	49407	-16129		

---

---

## Appendix C

---

---

# Important Firmware Locations

This appendix lists all significant firmware addresses: entry points, locations containing the addresses of entry points, and locations where machine and device identification bytes reside.

---

**Warning** The Monitor firmware entry points are the only *published* entry points in the sense that they are the only ones that will remain in the same locations in future Apple II series computers.

The firmware protocol identification bytes and offsets will work with other Apple II-series computers only if used as directed.

---

---

---

## The tables

This appendix supplements the chapter text by specifying three forms of each address: hexadecimal, decimal, and complementary (negative) decimal.

In these tables, some of the addresses are followed by a label. These labels are listed only to help you find the named location in the firmware listings, or to remember the function found at the address. The Apple IIc contains no global label table: your program must assign its own labels to the addresses as required.

There are several types of information at these firmware addresses: actual entry points (labeled *entry*), the low-order byte of an entry point (labeled *offset*), a device or machine identification byte (labeled *ident*), and indicators (labeled *indic*) specifying whether there are optional routines, vector addresses (labeled *vector*), or an RTS instruction location.

Each input/output port has an associated protocol table, as shown in Tables C-1 through C-4. Many of the bytes (labeled *offset*) in the protocol tables are the low-order bytes of addresses of I/O routines for the ports; the high-order byte of these addresses must be \$Cn (where n is the port number). This structure is explained in Chapter 3. Although your program must perform some extra processing to use these tables, the benefit is simplified compatible port and slot I/O for all Apple II-series machines.

---

---

## Port addresses

Addresses for serial ports 1 and 2, output port 3, and mouse input port 4 are shown in the following four tables.

**Table C-1**  
Serial port 1 addresses

Hex	Dec	Neg dec	Label	Type	Description
\$C100	49408	-16128		entry	Main port 1 entry point
\$C105	49413	-16123		ident	ID byte (\$38)
\$C107	49415	-16121		ident	ID byte (\$18)
\$C10B	49419	-16117		ident	Firmware card signature (\$01)
\$C10C	49420	-16116		ident	Super Serial Card ID (\$31)
\$C10D	49421	-16115		offset	Low-order PInit address
\$C10E	49422	-16114		offset	Low-order PRead address
\$C10F	49423	-16113		offset	Low-order PWrite address
\$C110	49424	-16112		offset	Low-order PStatus address
\$C111	49425	-16111		indic	Non-zero: no optional routines

**Table C-2**  
Serial port 2 addresses

Hex	Dec	Neg dec	Label	Type	Description
\$C200	49664	-15872		entry	Main port 2 entry point
\$C205	49669	-15867		iden	ID byte (\$38)
\$C207	49671	-15865		ident	ID byte (\$18)
\$C20B	49675	-15861		ident	Firmware card ID (\$01)
\$C20D	49676	-15860		ident	Super Serial Card ID (\$31)
\$C20D	49677	-15859		offset	Low-order PInit address
\$C20E	49678	-15858		offset	Low-order PRead address
\$C20F	49679	-15857		offset	Low-order PWrite address
\$C210	49680	-15856		offset	Low-order PStatus address
\$C211	49681	-15855		indic	Non-zero: no optional routines

**Table C-3**  
Video firmware addresses

Hex	Dec	Neg Dec	Label	Type	Description
\$C300	49920	-15616		entry	Main video entry point (output only)
\$C305	49925	-15611	C3KeyIn	ident	ID byte (\$38)
\$C307	49927	-15609	C3COut1	ident	ID byte (\$18)
\$C30B	49931	-15605		ident	Firmware card signature (\$01)
\$C30C	49932	-15604		ident	80-column card ID (\$88)
\$C30D	49933	-15603		offset	Low-order PInit address
\$C30E	49934	-15602		offset	Low-order PRead address
\$C30F	49935	-15601		offset	Low-order PWrite address
\$C310	49936	-15600		offset	Low-order PStatus address
\$C311	49937	-15599	MoveAux	entry	Routine for main/auxiliary control swapping (also called <i>AuxMove</i> )

**Table C-4**  
 Mouse port addresses

Hex	Dec	Neg dec	Label	Type	Description
\$C400	50176	-15360		entry	Main mouse entry point
\$C405	50181	-15355		ident	ID byte (\$38)
\$C407	50183	-15353		ident	ID byte (\$18)
\$C40B	50187	-15349		ident	Firmware card signature (\$01)
\$C40C	50188	-15348		type	X-Y pointing device ID (\$20)
\$C40D	50189	-15347		offset	Low-order PInit address
\$C40E	50190	-15346		offset	Low-order PRead address
\$C40F	50191	-15345		offset	Low-order PWrite address
\$C410	50192	-15344		offset	Low-order PStatus address
\$C411	50193	-15343		indic	Optional routines follow (\$00)
\$C412	50194	-15342	SetMouse	offset	Low-order SetMouse address
\$C413	50195	-15341	ServeMouse	offset	Low-order ServeMouse address
\$C414	50196	-15340	ReadMouse	offset	Low-order ReadMouse address
\$C415	50197	-15339	ClearMouse	offset	Low-order ClearMouse address
\$C416	50198	-15338	PosMouse	offset	Low-order PosMouse address
\$C417	50199	-15337	ClampMouse	offset	Low-order ClampMouse address
\$C418	50200	-15336	HomeMouse	offset	Low-order HomeMouse address
\$C419	50201	-15335	InitMouse	offset	Low-order InitMouse address

<b>Memory expansion</b>	The memory expansion version of the Apple IIc supports the mouse in port 7. This means that the firmware entry points are \$C7XX addresses, instead of \$C4XX address; change the 4's to 7's in Table C-4.
-------------------------	--

---

---

## Other video and I/O firmware addresses

Miscellaneous firmware addresses are listed in Table C-5.

**Table C-5**

Apple IIc enhanced video and miscellaneous firmware

Hex	Dec	Neg dec	Label	Type	Description
\$C600	50688	-14848	NewIRQ	entry	Disk drive firmware entry point
\$C700	50944	-14592		entry	External disk startup routine
\$C803	51203	-14333		entry	IRQ handling routine
<b>Memory expansion</b>			\$C700 supports the mouse in the memory expansion version.		

---

---

## Applesoft BASIC interpreter addresses

The addresses of Applesoft BASIC entry points are listed in the *Applesoft BASIC Programmer's Reference Manual*. The Applesoft interpreter occupies ROM addresses from \$D000 through \$F7FF.

---

---

## Monitor addresses

Table C-6 lists the Monitor entry points, machine identifier bytes, interrupt vectors, and the address of a known RTS instruction.

**Table C-6**

Apple IIc monitor entry points and vectors

Hex	Dec	Neg dec	Label	Type	Description
\$F800	63488	-2048	PLOT	entry	Plots a low-resolution block
\$F819	63513	-2023	HLine	entry	Draws low-resolution horizontal line
\$F828	63528	-2008	VLine	entry	Draws low-resolution vertical line
\$F832	63538	-1998	ClrScr	entry	Clears low-resolution screen
\$F836	63542	-1994	ClrTop	entry	Clears top 40 low-resolution lines
\$F864	63588	-1948	SetCol	entry	Sets low-resolution color (Table 5-4)
\$F871	63601	-1935	SCRN	entry	Reads color of low-resolution block
\$F941	63809	-1727	PrntAX	entry	Displays A and X in hex
\$F94A	63818	-1718	PrBl2	entry	Sends X blanks to output

**Table C-6** (continued)  
Apple IIc monitor entry points and vectors

Hex	Dec	Neg dec	Label	Type	Description
\$FA47	63845	-1691	NewBRK	entry	Apple IIc break handler
\$FA62	64098	-1438	Reset	entry	Hardware reset routine
\$FB1E	64386	-1150	PRead	entry	Reads hand controller position
\$FB6F	64467	-1169	SetPwrC	entry	Routine to create power-up byte
\$FBB3	64535	-1101		ident	Machine identification byte
\$FBC0	64548	-1088		ident	Machine identification byte
\$FBDD	64477	-1059	Bell1	entry	Sends 1-kHz beep to speaker
\$FC42	64578	-958	ClrEOP	entry	Clears from cursor to bottom
\$FC58	64600	-936	HOME	entry	Clears from cursor to upper left
\$FC9C	64668	-868	ClrEOL	entry	Clears from cursor to end of line
\$FC9E	64670	-866	ClEOLZ	entry	Clears from BASL to end of line
\$FCA8	64680	-856	WAIT	entry	Delays for time specified by A
\$FD0C	64780	-756	RdKey	entry	Displays cursor, jumps to KSW
\$FD1B	64795	-741	KeyIn	entry	Waits for keypress, reads key
\$FD35	64821	-715	RdChar	entry	Gets input, interprets ESC codes
\$FD67	64871	-665	GetLnZ	entry	Sends CR to output, goes to GetLn
\$FD6A	64874	-662	GetLn	entry	Displays prompt, gets input line
\$FD6F	64879	-657	GetLn1	entry	No prompt; gets input line
\$FD8B	64907	-629	CROut1	entry	Clears to end of line, calls CROut
\$FD8E	64910	-626	CROut	entry	Sends CR to output
\$FDDA	64986	-550	PrByte	entry	Sends A to output
\$FDE3	64995	-541	PrHex	entry	Displays low nibble of A in hex
\$FDED	65005	-531	COut	entry	Jumps to CSW
\$FDF0	65008	-528	COut1	entry	Displays A, advances cursor
\$FE2C	65068	-468	MOVE	entry	Copies memory elsewhere
\$FE36	65078	-458	VERIFY	entry	Compares two blocks of memory
\$FF2D	65325	-211	PrErr	entry	Sends ERR to output; beeps
\$FF3A	65338	-198	Bell	entry	Sends CONTROL-G to output
\$FF3F	65343	-193	IORest	entry	Loads \$45-\$49 into registers
\$FF4A	65354	-182	IOSave	entry	Stores A, X, Y, P, S at \$45-\$49
\$FF58	65368	-168	IORTS	RTS	Location of known RTS instruction
\$FF69	65385	-151	Monitor	entry	Standard Monitor entry point
\$FFFA	65530	-6		vector	Low-order NMI vector (unused)
\$FFFB	65531	-5		vector	High-order NMI vector (unused)
\$FFFC	65532	-4		vector	Low-order reset vector (\$62)
\$FFFD	65533	-3		vector	High-order reset vector (\$FA)
\$FFFE	65534	-2	IRQVect	vector	Low-order IRQ vector (\$03)
\$FFFF	65535	-1		vector	High-order IRQ vector (\$CB)

---

---

## Appendix D

---

---

# Operating Systems and Languages

This appendix is an overview of the characteristics of operating systems and languages when run on the Apple IIc. It is not intended to be a complete description. For more information, refer to the manuals that are provided with each product.

---

---

### Operating systems

This section discusses the operating systems that the Apple IIc works with CP/M, and any other operating system that requires an interface card, does not work on the Apple IIc.

---

### ProDOS

ProDOS is the preferred disk operating system for the Apple IIc. It supports startup from the external disk drive (on original Apple IIc's with the command PR#7), interrupts, and all other hardware and firmware features of the Apple IIc.

---

### DOS

The Apple IIc works with DOS 3.3. Its built-in disk drive hardware and firmware can also access DOS 3.2 disks by using the *BASIC*S disk. DOS support is provided for the sake of Apple II series compatibility; neither version of DOS takes full advantage of all the features of the Apple IIc.

---

## Pascal Operating System

Versions 1.2 and later of the Pascal Operating System use the 80/40 switch and the interrupt features of the Apple IIc, while remaining compatible with the other Apple II series computers.

While the Apple IIc works with Pascal 1.1, this version of the Pascal Operating System does not use the 80/40 switch or handle interrupts.

The Apple IIc does not work with Pascal 1.0, because the I/O firmware entry points of that version of the operating system are rigidly defined (rather than being accessed via a table), and the Apple IIc's built-in firmware does not correspond to these entry points.

---

---

## Languages

This section discusses using Apple programming languages with the Apple IIc. It is also a guide to using this reference manual with these languages.

---

### Applesoft BASIC

The programming examples in this manual are almost entirely in assembly language, and so most addresses and values are given in hexadecimal notation.

Use a PEEK in BASIC (instead of LDA in assembly language) to read a location, and a POKE (instead of STA) to write to a location. The values used by Applesoft must be in decimal, so you will have to convert hexadecimal values given in this manual to decimal. (Several tables in this manual include decimal equivalents to make the job easier for you.)

If you read a hardware address from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if a soft switch is on, its value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

---

## **Integer BASIC**

You will have to run a version of DOS in your Apple IIc to use Integer BASIC. ProDOS does not support Integer BASIC.

---

## **Pascal**

The Pascal language runs on the Apple IIc under versions 1.1 or later of the Pascal Operating System. However, for best performance, use Pascal versions 1.2 or later.

---

## **Fortran**

Fortran runs under version 1.1 of the Pascal Operating System, which does not detect or use certain Apple IIc features, such as the 80/40 switch or auxiliary memory. Therefore, Fortran does not take advantage of these features either.

---

## **Logo II**

Apple Logo II works under ProDOS on Apple II series machines with at least 128K of memory. Logo II is a version of the Logo language originally developed from the LISP (LISt Processing) language at MIT as a language to be used for learning. Logo II takes advantage of the Apple II's graphics and retains much of the power and flavor of LISP without LISP's somewhat cryptic syntax.



# Appendix E



## Interrupts

This appendix describes the sources of interrupts on the Apple IIc, how the firmware handles the interrupts, and how to use interrupt-driven features directly in those rare cases when the firmware cannot meet your needs.

---

**Warning** If you use Interrupt hardware directly, instead of using the built-in interrupt-handling firmware, you can't be sure that your programs will be compatible with possible future Apple II series computers or revisions.

---

---

### Introduction

This section describes interrupts and their effects on the Apple IIc hardware.

---

### What is an interrupt?

An **interrupt** is a signal that a computer uses to know when to stop what it's doing so it can quickly handle a time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This lets the system keep track of the mouse's position and maintain smooth movement of the pointer on the screen.

When an interrupt occurs, control passes to an interrupt handler, which must record the exact state of the computer at the moment of the interrupt, determine the source of the interrupt, and take appropriate action. It is important that the computer preserve a “snapshot” of its state when interrupted, so that when it continues later with what it was doing, those conditions can be restored.

---

## Interrupts on Apple II computers

Interrupts have not always been fully supported on the Apple II. All versions of Apple’s DOS, as well as the Monitor program, rely on the integrity of location \$45, which the built-in interrupt handler has always destroyed by saving the accumulator in it. Most versions of Pascal simply do not work with interrupts enabled.

The Apple IIc built-in interrupt handler now saves the accumulator on the stack instead of in location \$45. DOS 3.3, ProDOS, Pascal 1.2 (or later versions), and the Monitor all work with interrupts on the Apple IIc.

You should use either ProDOS or Pascal 1.2 (or later versions) if you want interrupt-using software to work on the Apple IIe and the Apple II Plus. Both operating systems have full interrupt support built in.

Interrupts are effective only if they are enabled most of the time since interrupts that occur while interrupts are disabled cannot be detected. Because of the critical timing of disk read and write operations, Pascal, DOS 3.3, and ProDOS turn off interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all the interrupt sources discussed below are turned off.

On the Apple IIe only, interrupts are periodically turned off while 80-column screen operations are being performed. This is most noticeable while the screen is scrolling. Also, most peripheral cards used in the Apple IIe disable interrupts while reading and writing.

---

## Interrupt handling on the 65C02

From the point of view of the 65C02, there are three possible causes of interrupts.

1. The IRQ line on the microprocessor can be pulled low if 65C02 interrupts are not masked (that is, the CLI instruction has been used). This is the standard technique that devices use when they need immediate attention.
2. The processor executes a break (BRK, opcode \$00) instruction.
3. A nonmaskable interrupt (NMI) occurs. Because the NMI line in the Apple IIc's 65C02 is not used, this never happens on the Apple IIc.

The first two possibilities cause the 65C02 to save the current program counter and status byte on the stack and then jump to the routine whose address is stored in \$FFFE and \$FFFF. The sequence performed by the 65C02 is:

1. If an IRQ occurs, finish executing the current instruction. (If a BRK occurs, the current instruction is already finished.)
2. Push the high byte of the program counter onto the stack.
3. Push the low byte of the program counter onto the stack.
4. Push the program status byte onto the stack.
5. Jump to the address stored in \$FFFE, \$FFFF—that is, JMP (\$FFFE).

The different sources of interrupt signals are discussed below.

---

## The interrupt vector at \$FFFE

In the Apple IIc there are three separate regions of memory that contain address \$FFFE: the built-in ROM, the bank-switched memory in main RAM, and the bank-switched memory in auxiliary RAM. The vector at \$FFFE in the ROM points to Apple IIc's built-in interrupt handling routine. You should generally use the built-in interrupt handler, rather than writing your own, because of the complexity of interrupts on the Apple IIc.

When you initialize the mouse or serial communication firmware, copies of the ROM's interrupt vector are placed in the interrupt vector addresses in both main and auxiliary bank-switched memory. If you plan to use interrupts and the bank-switched memory without the mouse or communication firmware, you must copy the ROM's interrupt vector yourself.

---

---

## The built-in interrupt handler

The built-in interrupt handler is responsible for determining whether a BRK or an IRQ interrupt occurred. If it was an IRQ interrupt, it decides whether the interrupt should be handled internally, handled by the user, or simply ignored.

The built-in interrupt-handling routine records the current memory configuration, then sets up its own standard memory configuration so that a user's interrupt handler knows the precise memory configuration when it is called.

Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described later in this appendix.

If the interrupt was not caused by a BRK, the handler checks for interrupts that it knows how to handle (for example, a properly initialized mouse) and handles them.

Depending on the state of the system, it either ignores other interrupts or passes them to a user's interrupt handling routine whose address is stored at \$03FE and \$03FF of main memory.

After handling an interrupt itself, or after the user's handler returns (with an RTI), the built-in interrupt handler restores the memory configuration, and then does an RTI to restore processing to where it was when the interrupt occurred. Table E-1 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

**Table E-1**  
Interrupt-handling sequence

Interrupted program	Processor	Built-in handler	User's handler
Program →	Push address Push status JMP (\$FFFE) →	Save old and set new memory configuration .. If BRK, then go to break handler (\$FA47) →	
		Our interrupt? NO: Push address Push status JMP (\$03FE) →	Handle interrupt ...
		YES: Handle it	...
		Restore memory configuration ← RTI	
	Pull status ← RTI		
Program ←	Pull address		

## Saving the memory configuration

The built-in interrupt handler saves the state of the system, and sets it to a known state according to these rules:

- ☐ If 80Store and Page2 are on, then it switches in text Page 1 (Page2 off) so that main screen holes are accessible.
- ☐ It switches in main memory for reading (RAMRd off).
- ☐ It switches in main memory for writing (RAMWrt off).
- ☐ It switches in ROM addresses \$D000–\$FFFF for reading (RdLCRAM off).
- ☐ It switches in main stack and zero page (AltZP off).
- ☐ It preserves the auxiliary stack pointer, and restores the main stack pointer.

- It preserves the current ROM state and switches in the ROM bank 1.
- ❖ *Note:* Because main memory is switched in, all memory addresses used later in this appendix are in main memory unless otherwise specified.

---

## Managing main and auxiliary stacks

Because the Apple IIc has two stack pages, the firmware has established a convention that allows the system to be run with two separate stack pointers. Two bytes in the auxiliary stack page are to be used as storage for inactive stack pointers: \$0100 for the main stack pointer when the auxiliary stack is active, and \$0101 for the auxiliary stack pointer when the main stack is active.

When a program that uses interrupts switches in the auxiliary stack for the first time, it should place the value of the main stack pointer at auxiliary stack address \$0100, and initialize the auxiliary stack pointer to \$FF (the top of the stack). When it subsequently switches from one stack to the other, it should save the current stack pointer before loading the pointer for the other stack.

When an interrupt occurs while the auxiliary stack is switched in, the current stack pointer is stored at \$0101, and the main stack pointer is retrieved from \$0100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.

---

---

## User's interrupt handler at \$03FE

You can set up screen hole locations to indicate that the user's interrupt handler should be called when certain interrupts occur. To do this, place your interrupt handler's address at \$03FE and \$03FF in main memory, low byte first.

The user's interrupt handler should do the following:

- Verify that the interrupt came from the expected source. The following sections describe how this should be done for each interrupt source.
- Handle the interrupt as desired.
- Clear the interrupt, if necessary. The following sections describe how to clear the interrupts.
- Return with an RTI.

If your interrupt handler needs to know the memory configuration at the time of the interrupt, it can check the encoded byte stored four bytes down on the stack. This byte is explained later in this appendix.

In general there is no guaranteed *maximum* response time for interrupts. This is because the system may be doing a disk operation, which could last for several seconds.

Once the built-in interrupt handler has been called, it takes about 250 to 300 microseconds for it to call your interrupt-handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.

If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst possible state (80Store and Page2 on, auxiliary memory switched in for reading and writing, auxiliary bank-switched memory page \$02 switched in for reading and writing).

---

---

## Handling break instructions

After the interrupt handler has set the memory configuration, it checks to see if the interrupt was caused by a BRK (opcode \$00) instruction. (If it was, bit 4 of the processor status byte is a 1.) If so, it jumps to a break-handling routine, which saves the state of the computer at the time of the break as follows:

Information	Location
Program counter (low byte)	\$3A
Program counter (high byte)	\$3B
Encoded memory state	\$44
Accumulator	\$45
X register	\$46
Y register	\$47
Status register	\$48

Finally, the break routine jumps to the routine whose address is stored at \$03F0 and \$03F1.

The encoded memory state in location \$44 can be interpreted as follows:

Bit 7 = 0

Bit 6 = 1 if 80Store and Page2 both on

Bit 5 = 1 if auxiliary RAM switched in for reading

Bit 4 = 1 if auxiliary RAM switched in for writing

Bit 3 = 1 if bank-switched RAM being read

Bit 2 = 1 if bank-switched \$D000 page \$01 switched in

Bit 1 = 1 if bank-switched \$D000 page \$02 switched in

Bit 0 = 0

---

---

## Sources of interrupts

The Apple IIc can receive interrupts from many different sources. Each source is enabled and used slightly differently from the others. There are two basic sources of interrupts: use of the mouse, and actions affecting the two 6551 ACIA circuits (the chips that control serial communication). How to use these sources of interrupts in conjunction with the built-in interrupt handler is discussed later in this appendix.

Mouse use can cause interrupts when

- ☐ the mouse is moved in the horizontal (X) direction
- ☐ the mouse is moved in the vertical (Y) direction
- ☐ the mouse button is pressed

Interrupts can also be generated every 1/60 second by the rising edge of the vertical blanking signal. This is called the *vertical blanking* (VBL) interrupt and is synchronized with a signal used for the video display.

Actions affecting the ACIA circuits can cause interrupts when

- ☐ a key is pressed (the firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user)
- ☐ either ACIA has received a byte of data from its port (the firmware can use this interrupt to buffer data or it can pass the interrupt on to the user)
- ☐ pin 5 of either serial port changes state (device ready/not ready to accept data) (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)

- either ACIA is ready to accept another character to be transmitted (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)
- the keyboard strobe is cleared (the firmware absorbs this interrupt)

An interrupt can also be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.

---

---

## Firmware handling of interrupts

The following sections discuss how the various sources of interrupts should be used together with the built-in interrupt handler.

---

### Firmware for mouse and VBL

As described in Chapter 9, the mouse can be initialized (by the SetMouse call) to nine different modes that enable one or more sources of interrupts. In transparent mode, the interrupts are entirely handled by the built-in interrupt handler; the other modes require a user-installed interrupt handler.

When the mouse is initialized, the interrupt vector is copied to addresses \$FFFE and \$FFFF in main and auxiliary bank-switched RAM. This permits mouse interrupts with any memory configuration.

When the mouse is active, possible sources of interrupts are those listed earlier in this appendix as resulting from mouse use.

When an interrupt occurs, the built-in interrupt handler determines whether that particular interrupt source was enabled (by the SetMouse call). If so, the user's interrupt handler, whose address is stored at \$03FE, is called.

The user's interrupt handler should first call ServeMouse to determine the source of the interrupt. This call updates the mouse status byte at \$077C and returns with the carry bit clear if mouse movement, button, or vertical blanking was the source of the interrupt.

The values of this mouse status byte at \$077C (\$077F in the memory expansion IIc) are as follows:

**Bit    1 means that**

- 3    Interrupt was from vertical blanking
- 2    Interrupt was from button
- 1    Interrupt was from mouse movement

If the interrupt was due to mouse movement or button, the user's interrupt handler should then do a call to ReadMouse. This causes the mouse coordinates and status to be updated as follows:

- \$047C    Low byte of X coordinate
- \$04FC    Low byte of Y coordinate
- \$057C    High byte of X coordinate
- \$05FC    High byte of Y coordinate
- \$077C    Button and movement status

**Bit    Means**

- 7    0 = button up; 1 = button down
- 6    0 = button up on last ReadMouse  
     1 = button down on last ReadMouse
- 5    0 = no movement since last ReadMouse  
     1 = movement since last ReadMouse
- 3-1   Always set to 0 (interrupt cleared)

After the interrupt has been handled, the routine should terminate with an RTI.

Remember that interrupts may be missed during disk accesses.

If you turn on mouse interrupts without initializing the mouse, the built-in interrupt handler will absorb the interrupts. If you want to handle mouse interrupts yourself, you must write your own interrupt handler and place vectors to it at addresses \$FFFE and \$FFFF in bank-switched RAM. Interrupts will be ignored whenever the \$D000-\$FFFF ROM is switched in.

---

## Firmware for keyboard interrupts

The Apple IIc hardware is able to generate an interrupt when a key is pressed. The firmware is able to buffer up to 128 keystrokes, completely transparently, when properly enabled to do so. It saves them in the second half of page \$08 of auxiliary memory. After the buffer is full, subsequent keystrokes are ignored. Because interrupts are only generated when keypresses occur, characters generated by the auto-repeat feature are not buffered. They can, however, be read when the buffer is empty.

Once keyboard buffering has been turned on, the next key should be read by calling RdKey (\$FD0C).

---

**Warning** Do not call the buffer reading routine directly. Its entry address will not be the same in future versions of the computer.

---

The special characters Control-S (stop list) and Control-C (stop Applesoft execution) do not work while keyboard buffering is turned on. A new keystroke, Solid Apple-Control-X, clears the buffer.

### Using keyboard buffering firmware

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself this way:

---

**Memory expansion** The Apple IIc that supports memory expansion places the keyboard screen holes in different locations from those used in earlier versions. For the memory expansion IIc, change all \$nnnF addresses to \$nnnC (that is, change \$05FF to \$05FC).

---

1. Disable processor interrupts (SED).
2. Set location \$05FA to \$80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations \$05FF and \$06FF to \$80. These are pointers to where in the buffer the next keystroke will be stored and where the next will be read from, respectively.
4. Turn on the ACIA for port 2 by setting the low nibble of \$C0AA to the value \$0F. For example:

```
LDA $C0AA    Read port 2 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C0AA    Set port 2 ACIA command register
```

If you are using the serial ports at the same time, just set the low bit of \$C0AA to 1. This prevents receiver interrupts from being turned off.

A PR#2 or IN#2 or the equivalent will shut off keyboard interrupts.

5. Enable processor interrupts (CLI).

## Using keyboard interrupts through firmware

Keyboard interrupts are received through the ACIA for port 2. They can be enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location \$05FA to \$C0. This tells the firmware to identify a keystroke interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 2 by setting the low nibble of \$C0AA to the value \$0F. For example:

```
LDA $C0AA    Read port 2 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C0AA    Set port 2 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify the keyboard as the interrupt source by reading location \$04FA. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 2, bit 7 is set. If the interrupt was caused by a keystroke, bit 6 is set and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting \$04FA to \$00.

---

## Using external interrupts through firmware

Pin 9 of the external disk drive connector (EXTINT) can be used to generate interrupts through the ACIA for port 1. It can be used as a source of interrupts (on a high-to-low transition) if enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location \$05F9 to \$C0. This tells the firmware to identify an external interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 1 by setting the low nibble of \$C09A to the value \$0F. For example:

```
LDA $C09A    Read port 1 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C09A    Set port 1 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify this interrupt by reading location \$04F9. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 1, bit 7 is set. If the interrupt was caused by the external interrupt line, bit 6 is clear and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting \$04F9 to \$00.

---

## Firmware for serial interrupts

The Apple IIc hardware is able to generate interrupts both when the ACIA receives data and when it is ready to send data. The built-in interrupt handler responds to incoming data only. The firmware is able to buffer up to 128 incoming bytes of serial data from either serial port. After the buffer is full, data are ignored. Only one port can be buffered at a time. The following sections assume that the serial port to be buffered is already initialized, as explained in Chapter 8.

### Using serial buffering transparently

Serial buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself, as follows:

---

#### Memory expansion

For the memory expansion IIc, change all \$nnnF addresses to \$nnnC and change the \$0D value to \$09.

---

1. Disable processor interrupts (SEI).
2. Set location \$04FF to \$C1 to buffer port 1, or to \$C2 to buffer port 2.
3. Set locations \$057F and \$067F to \$00. These are pointers to the next byte in the buffer to be used and the next character to be read from the buffer, respectively.
4. Turn on the ACIA for the port by setting the low nibble of \$C09A for port 1 or \$C0AA for port 2 to \$0D. For example:

```
LDA $C09A    Read port 1 ACIA command register
AND $F0      Clear low nibble
ORA #$0D     Set low nibble to $0D
STA $C09A    Set port 1 ACIA command register
```

The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

## 5. Enable processor interrupts (CLI).

When serial port buffering is thus enabled, normal reads from the serial port firmware fetch data from the buffer rather than directly from the ACIA.

### Using serial interrupts through firmware

It is also possible to use the firmware to call the user interrupt handler whenever a byte of data is read by the ACIA. In this mode buffering is not performed by the firmware.

---

#### Memory expansion

For the memory expansion llc, change all \$nnnF addresses to \$nnnC and change the \$0D value to \$09.

---

1. Disable processor interrupts (SEI).
2. Set location \$04FF to a value other than \$C1 or \$C2.
3. Turn on the ACIA for the port by setting the low nibble of \$C09A for port 1 or \$C0AA for port 2 to \$0D. For example:

```
LDA $C09A    Read port 1 ACIA command register
AND $F0      Clear low nibble
ORA #$0D     Set low nibble to $0D
STA $C09A    Set port 1 ACIA command register
```

The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

## 4. Enable processor interrupts (CLI).

When a serial port is thus enabled, the user's interrupt handler is called each time the port receives a byte of data. The status byte saved by the firmware (\$04F9 for port 1; \$04FA for port 2) has the high bit set if the interrupt occurred on that port. Bit 3 is set if the interrupt was due to a received byte of data.

The interrupt handler should clear the interrupt by clearing bits 7 and 3 of that port's status byte (\$04F9 for port 1; \$04FA for port 2).

### Transmitting serial data

The serial firmware does not implement buffering for serial output. Instead it waits for two conditions to be true before transmitting a character:

- ☐ The ACIA's transmit register must be ready to accept a character. This is true if bit 4 of the ACIA's status register is 1.
- ☐ The device must signal that it is ready to accept data. This is true if bit 5 of the ACIA's status register is 0. Bit 5 is 0 if pin 5 of the port's connector is also 0.

When the serial firmware is active, a change of state on pin 5 of that port generates an interrupt. That interrupt is absorbed, but the data remain in bit 5 of the status register. Interrupts from the ACIA's transmit register are normally disabled.

### **A loophole in the firmware**

So that programs can make use of interrupts on the ACIAs without affecting mouse interrupt handling, there is a tiny loophole purposely left in the built-in interrupt handler. If transmit interrupts are enabled on the ACIA—that is, if bits 3, 2, and 0 of the ACIA's command register have the values 0, 1, and 1, respectively—then control is passed to the user's interrupt handler if the interrupt is not intended for the mouse (movement, button, or VBL).

This means that you can write more sophisticated serial interrupt-handling routines than the limited firmware space could provide (such as printer spooling). The firmware will still set memory to its standard state, handle mouse interrupts, and restore memory after your routine is finished.

When you receive the interrupt, neither ACIA's status register has been read. You are fully responsible for checking for interrupts on both ACIAs, determining which of the four interrupt sources on each ACIA caused the interrupt, and how to handle them. Refer to the 6551 specification for more details. The built-in firmware itself is an excellent example of how interrupts on the ACIA can be handled.

---

---

## **Bypassing the interrupt firmware**

The following sections give further details on using interrupts on the Apple IIc computer without using the built-in interrupt handler.

A method of handling mouse interrupts directly is described in Chapter 9.

---

### **Using mouse interrupts without the firmware**

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If the \$D000–\$FFFF ROM is ever switched in, the built-in interrupt handler will absorb the mouse interrupts.

Tables E-2 and E-3 show how to activate and read mouse interrupts without using the firmware. Remember to disable interrupts (SEI) before enabling mouse interrupts, then turn them on when done (CLI).

**Table E-2**  
Activating mouse interrupts

To activate interrupts on	Enable IOU access	Select source	Enable source	Disable IOU access
Mouse X (rising edge)	STA \$C079	STA \$C05C	STA \$C059	STA \$C078
Mouse X (falling edge)	STA \$C079	STA \$C05D	STA \$C059	STA \$C078
Mouse Y (rising edge)	STA \$C079	STA \$C05E	STA \$C059	STA \$C078
Mouse Y (falling edge)	STA \$C079	STA \$C05F	STA \$C059	STA \$C078
VBL	STA \$C079		STA \$C05B	STA \$C078

**Table E-3**  
Reading mouse interrupts

To read interrupts from	Read direction (A.S.A.P)	Determine source	Handle it	Return
Mouse X	LDA \$C066	LDA \$C015 (bit 7=1 if true)	...	RTI
Mouse Y	LDA \$C067	LDA \$C017 (bit 7=1 if true)	...	RTI
VBL		LDA \$C019 (bit 7=1 if true)	...	RTI

The mouse direction data read from \$C066 and \$C067 are guaranteed valid for at least 40 microseconds, and average duration is at least 200 microseconds, so you should read the direction as soon as possible.

---

## Using ACIA interrupts without the firmware

To use ACIA interrupts without the firmware, you must set your own interrupt vectors. If the \$D000–\$FFFF ROM is ever switched in, the built-in interrupt handler will handle the interrupt as determined by certain mode bytes.

When writing your serial interrupt handler, refer to Figures 11-31 through 11-33 and to the Synertek 6551 ACIA specification. As shown in Chapter 11, the ACIAs have the following connections:

- Port 1**     DSR line connected to the EXTINT line on the external disk port.  
              DCD line connected to pin 5 of port 1 connector.
- Port 2**     DSR line goes high when a key is pressed.  
              DCD line connected to pin 5 of port 2 connector.

The ACIA registers have the following addresses:

<b>Port 1</b>		<b>Port 2</b>	
Data register	= \$C098	Data register	= \$C0A8
Status register	= \$C099	Status register	= \$C0A9
Command register	= \$C09A	Command register	= \$C0AA
Control register	= \$C09B	Control register	= \$C0AB



## Appendix F



# Apple II Series Differences

This appendix compares the Apple IIc to the Apple IIe, Apple II Plus, and Apple II. It does not contain an exhaustive list of differences, but it does mention those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more Apple II series models.

---

---

### Overview

The differences between the Apple II series computers can be expressed as a series of equations: this computer equals that one plus or minus certain features.

The following equations compare each model of Apple II series with its predecessor in terms of functional equivalence, not literal equality. For example,

**Apple II Plus = Apple II – Integer BASIC firmware**

does not mean that Integer BASIC firmware can be removed from the Apple II—just that the one machine functions as if it were the other without such firmware.

**Apple II Plus = II**

- + Autostart ROM
- + Applesoft firmware
- + 48K RAM standard
- old Monitor ROM
- Integer BASIC firmware

- Apple IIe** = II Plus + Apple Language Card (with 16K of RAM)
- + 80-column (enhanced) video firmware
  - + built-in diagnostics
  - + full ASCII keyboard
  - + internal power light
  - + FCC approval
  - + improved back panel
  - + 9-pin back panel game connector
  - + auxiliary slot (with possibility of 80-column text card and extra 64K RAM)
  - slot 0
  - + interrupt support in firmware (enhanced Apple IIe)
  - + Mini-Assembler in firmware (enhanced Apple IIe)
- Apple IIc** = IIe
- + extended 80-column text card
  - + 80/40 switch
  - + keyboard switch
  - + disk-use light
  - + disk controller port
  - + disk drive
  - + mouse port
  - + serial printer port
  - + serial communication port
  - + built-in port firmware
  - + video expansion connector
  - removable cover
  - slots 1 to 7
  - auxiliary slot
  - internal power light
  - cassette I/O connectors
  - internal game I/O connector (hence no game output)
  - auxiliary video pin
  - Monitor cassette support
  - + Mini-Assembler in firmware (Apple IIc with UniDisk 3.5 support)
  - + Smartport in firmware (UniDisk 3.5 and memory expansion Apple IIc)
  - + memory expansion card support (memory expansion Apple IIc)

---

## Type of processor

The processor in the Apple II and II Plus is the 6502. The original Apple IIe uses a 6502A. The Apple IIc and enhanced Apple IIe both use the 65C02: this is a redesigned CMOS CPU that has 27 new instructions, new addressing modes, and for some instructions a differing execution scheme and machine cycle counts (see Appendix A).

Programs written for the Apple IIc will run on the earlier machines only if they do not contain instructions unique to the 65C02, or depend on shared instructions whose cycle times differ. Programs should also use only published entry points in the Monitor firmware to allow maximum compatibility between different Apple II series computers.

---

## Machine identification

Identification of Apple II series computers is as shown in Table F-1.

**Table F-1**  
Apple II series identification bytes

Machine	\$FBB3	\$FB1E	\$FBC0	\$FBBF
Apple II	\$38			
Apple II Plus	\$EA	\$AD		
Apple IIe	\$06		\$EA	
Apple IIe (enhanced)	\$06		\$E0	
Apple IIc	\$06		\$00	\$FF
Apple IIc (UniDisk 3.5 support)	\$06		\$00	\$00
Apple IIc (memory expansion)	\$06		\$00	\$03
Apple III in Apple II emulation mode	\$EA	\$8A		

Any future Apple II series computer or ROM release will have different values in these locations. Machine identification routines are available from Apple Vendor Technical Support.

The MachID byte for ProDOS (\$BF98 on the global page) will have bit 3 set to 0 if the computer is an Apple II, II Plus, IIe, or III, and to 1 if the computer is not one of these machines. In an Apple IIc, bits 7 and 6 are also set to binary 10.

Bits 7 and 6 set to binary 10 indicate that a computer is Apple IIe and IIc compatible, regardless of the value of bit 3.

---

---

## Memory structure

This section compares the memory organization of the Apple IIc with that of the Apple II, II Plus, and IIe. These machines differ in RAM space, ROM space, slot or port address space, and hardware page use.

---

### Amount and address ranges of RAM

The Apple II could have as little as 4K of RAM at the time of purchase, and could be upgraded to as much as 48K of RAM.

The Apple II Plus has 48K of RAM (\$0000 through \$BFFF) as a standard feature. With the addition of an Apple Language Card, a 48K Apple II or II Plus could be expanded to have 64K of RAM.

The Apple IIe has a full 64K of RAM. The top 12K addresses overlap with the ROM addresses \$D000 through \$FFFF. There is an additional bank-switched area of 4K from \$D000 through \$DFFF. This arrangement is equivalent to an Apple II Plus with an Apple Language Card installed. A program selects between the RAM and ROM address spaces and between the \$Dxxx banks by changing soft switches located in memory.

With an Extended 80-Column Text Card installed in its auxiliary slot, an Apple IIe has an additional 64K of RAM available, although no more than half of the 128K of RAM space is available at any given time. Soft switches located in memory control these address space selections.

The RAM in the Apple IIc is equivalent to the RAM in an Apple IIe with an Extended 80-Column Text Card. The optional memory expansion card can add as much as 1Mb of RAM to the IIc in 256K steps.

---

### Amount and address ranges of ROM

The Apple II has 8K of ROM (\$E000 through \$FFFF), and the Apple II Plus has 12K of ROM (\$D000 through \$FFFF). Users can plug their own ROMs into the sockets provided. The on-board (as opposed to slot) ROM address range is from \$D000 through \$FFFF.

The Apple IIe has 16K of ROM, of which it uses 15.75K (addresses \$C100 through \$FFFF; page \$C0 addresses are for I/O hardware). ROM addresses \$C300 through \$C3FF (normally assigned to the ROM in a card in slot 3) and \$C800 through \$CFFF contain 80-column video firmware; ROM addresses \$C100 through \$C2FF and \$C400 through \$C7FF (normally assigned to the ROM on cards in slots 1, 2, 4, 5, 6, and 7) contain built-in self-test routines.

A soft switch in RAM controls whether the video firmware or slot 3 card ROM is active. Invoking the self-tests with Solid Apple-Control-Reset causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc ROM also uses the 15.75K from \$C100 through \$FFFF, and its enhanced video firmware has the same entry point addresses as on the Apple IIe. However, there are only rudimentary built-in self-tests, and these do not preempt any port firmware space.

---

### UniDisk 3.5

The Apple IIc with built-in UniDisk 3.5 support has twice the ROM (32K) of the original Apple IIc. The extra ROM contains support for the Smartport, a Mini-Assembler, STEP and TRACE functions in the Monitor firmware, expanded self-test routines, and improved interrupt support.

---

In the Apple IIc, addresses \$C100 through \$CFFF contain I/O and interrupt firmware, addresses \$D000 through \$F7FF contain the Applesoft BASIC interpreter, and addresses \$F800 through \$FFFF contain the Monitor.

---

## Peripheral-card memory spaces

Each Apple IIc port has up to 16 peripheral-card I/O space locations in main memory on the hardware page (beginning at location \$C0s0 + \$80 for slot or port s), allocated in the standard Apple II series way (that is, beginning at location \$C0s0 + \$80 for each slot s).

The peripheral-card ROM space (page \$Cs for slot s in the Apple II, II Plus, and IIe) contains the starting and entry-point addresses for port s, but port routines are not limited to their allocated \$Cs pages.

The 2K-byte expansion ROM space from \$C800 to \$CFFF in the Apple IIc is used by the enhanced video firmware and miscellaneous I/O and memory-transfer routines.

The 128 bytes of peripheral-card RAM space (or scratch-pad RAM) (64 screen holes in main memory and their equivalent addresses in auxiliary memory) are reserved for use by the built-in firmware. It is extremely important for the correct operation of Apple IIc firmware that these locations not be altered by software except for the specific purposes described in Chapters 7, 8, and 9, and in Appendix E.

---

## Hardware addresses

The hardware page (the addresses from \$C000 through \$C0FF) controls memory selection and input/output hardware characteristics. All input and output (except video output) takes place at one or more hardware page addresses. For the sake of simplicity, this section presents only a general comparison between the Apple IIc on the one hand, and the Apple II, II Plus, and IIe on the other, with respect to hardware page use. However, for many characteristics, the Apple IIe and IIc work one way, while the Apple II and II Plus work another.

### **\$C000-\$C00F**

On all Apple II series computers, reading any one of these addresses reads the keyboard data and strobe. On the Apple IIe and IIc, writing to each of these addresses turns memory and display switches on and off. Writing to addresses \$C006, \$C007, \$C00A, and \$C00B performs ROM selection on the Apple IIe. Writing to these four addresses is reserved on the Apple IIc.

For reading the keyboard, use \$C000; reserve \$C001 through \$C00F.

### **\$C010-\$C01F**

On all Apple II series computers, writing to any one of these addresses clears the keyboard strobe. On the Apple IIe and IIc, reading each of these addresses checks the status of a memory or display switch, or the any-key-down flag.

For clearing the keyboard strobe, use \$C010; reserve \$C011 through \$C01F.

Reading \$C015 checks the SLOTCXROM switch on the Apple IIe, but it resets the X-movement interrupt (XInt) on the Apple IIc. Similarly, reading \$C017 checks the SLOTC3ROM switch on the Apple IIe, but it resets the Y-movement interrupt (YInt) on the Apple IIc.

Reading \$C019 checks the current state of vertical blanking (VBL) on the Apple IIe, but it resets the latched vertical blanking interrupt (VBIInt) on the Apple IIc.

### **\$C020-\$C02F**

On the Apple II, II Plus, and IIe, reading any address \$C02x toggles the cassette output signal. On the original Apple IIc, both reading from and writing to these locations are reserved. The Apple IIc with 32K of ROM uses \$C028 to switch in or out the extra 16K of ROM.

### **\$C030-\$C03F**

On all Apple II series computers, reading an address of the form \$C03x toggles the speaker. For full Apple II series compatibility, toggle the speaker using \$C030, and reserve \$C031 through \$C03F.

On the Apple IIc, writing to \$C031 through \$C03F is explicitly reserved.

### **\$C040-\$C04F**

On the Apple II, II Plus, and IIe, reading any address of the form \$C04x triggers the utility strobe. The Apple IIc has no utility strobe.

On the Apple IIc, addresses \$C044 through \$C047 are explicitly reserved, and reading or writing any address from \$C048 through \$C04F resets both the X and Y mouse interrupts (XInt and YInt).

### **\$C050-\$C05F**

Addresses \$C050 through \$C057 work the same on the Apple IIc as on the Apple IIe: they turn the TEXT, MIXED, Page2, and HiRes switches on and off.

On the Apple IIe, addresses \$C058 through \$C05F turn the annunciator outputs on and off. On an Apple IIe with a revision B main logic board or later, an Apple Extended 80-Column Text Card, and a jumper installed on the card, reading locations \$C05E and \$C05F set and clear double high-resolution display mode.

On the Apple IIc, if the IOUDis switch is on, both reading from and writing to addresses \$C058 through \$C05D are reserved, and addresses \$C05E and \$C05F set and clear the double high-resolution display (as on the Apple IIe equipped as described in the preceding paragraph). If the IOUDis switch is off, then addresses \$C058 through \$C05F control various characteristics of mouse and vertical blanking interrupts (Table 9-2).

### **\$C060-\$C06F**

On the Apple IIc, writing to any address of the form \$C06x is reserved, and reading addresses \$C068 through \$C06F is reserved.

Reading addresses \$C061 and \$C062 is the same as on the Apple IIe (switch inputs and Apple keys). Reading addresses \$C064 and \$C065 is the same as on all other Apple II series computers (analog inputs 0 and 1).

On the Apple IIc, address \$C063 bit 7 is 1 if the mouse switch is not pressed, and 0 if it is pressed, so that software looking for the shift-key mod (used on Apple II, II Plus, and IIe with some text cards) will find it and display lowercase correctly. If by chance the mouse button is pressed when the software checks location \$C063, it will appear that the Shift-key mod is not present.

On the Apple IIc, address \$C060 is used for reading the state of the 80/40 switch; on the Apple II, II Plus, and IIe, this address is for reading cassette input.

The Apple IIc has two, rather than four, analog (paddle) inputs. Addresses \$C066 and \$C067 are used for reading the mouse X and Y direction bits.

### **\$C070-\$C07F**

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form \$C07x triggers the (analog input) paddle timers.

On the Apple IIc, only address \$C070 is to be used for that one function. Addresses \$C071 through \$C07D are explicitly reserved. The results of reading from or writing to addresses \$C07E and \$C07F are described in Table 5-8.

### **\$C080-\$C08F**

On the Apple IIe and IIc, accessing addresses in this range selects different combinations of bank-switched memory banks. However, addresses \$C084 through \$C087 duplicate the functions of the four addresses preceding them, and addresses \$C08C through \$C08F do also. These eight addresses are explicitly reserved on the Apple IIc.

### **\$C090-\$C0FF**

On the Apple II, II Plus, and IIe, each group of 16 addresses of the form \$C080 + \$s0 is allocated to an interface card (if present) in slot s.

On the Apple IIc, addresses corresponding to slots 1, 2, 3, 4, and 6 are allocated to a serial interface card, communication interface card, 80-column text card, mouse interface card, and disk controller card, respectively. All other addresses in this range are reserved.

---

## **Monitors**

The older models of the Apple II and Apple II Plus included a different version of the System Monitor from the one built into more recent models (and the Apple IIe and IIc). The older version, called the Monitor ROM, had the same standard I/O subroutines as the newer Autostart ROM, but a few of their features were different; for example, there were no arrow keys for vertical cursor motion.

When you start the Apple IIc with a DOS or *BASIC*S disk and it loads Integer BASIC into the bank-switched area in RAM, it loads the old Monitor along with it. When you type INT from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type FP to switch back to Applesoft, which uses the new Monitor in ROM, or activate the 80-column firmware.

---

---

## I/O in general

Apple IIc I/O is different from I/O on the Apple II, II Plus, and IIe in three important respects: the possibility of direct memory access (DMA) transfers, the presence or absence of slots, and the presence or absence of built-in interrupt handling.

---

## DMA transfers

The Apple II, II Plus, and IIe allow DMA transfers, because both the address and the data bus are available at the slots. No true DMA transfer is possible with the Apple IIc because neither bus is available at any of the back panel connectors.

---

## Slots versus ports

The Apple II and II Plus have eight identical slots; the Apple IIe has seven identical slots plus a 60-pin auxiliary slot for video, add-on memory, and test cards. The Apple IIc has no slots; instead, it has back panel connectors and built-in hardware and firmware that are functional equivalents of slots with cards in them. The back panel connectors are called *ports* on the Apple IIc.

---

## Interrupts

The Apple IIc is the first computer in the Apple II series to have built-in interrupt-handling capabilities. The enhanced Apple IIe has very similar interrupt-handling capability included.

---

---

## The keyboard

Both keyboard layout and character sets vary in the Apple II series computers. The major keyboard difference in the Apple II series is that the Apple IIe and IIc have full ASCII keyboards, while the Apple II and II Plus do not.

---

## Keys, switches, and lights

The Apple II and II Plus have identical 52-key keyboards. The Apple IIe and Apple IIc keyboards have the same 63-key full ASCII keyboard layout, with new and repositioned keys and characters as compared to the Apple II and II Plus. While the Apple II and II Plus have a Rept key, the IIe and IIc have an auto-repeat feature built into each character key.

Some Apple II and Apple II Plus machines have a slide switch inside the case, under the keyboard edge of the cover, for selecting whether or not Reset works without Control. On the Apple IIe and Apple IIc, there is no choice: Control-Reset works, and Reset alone does not.

The Apple IIc and IIe have an Open Apple and a Solid Apple key; the Apple II and II Plus do not have these two keys.

The captions on several keys—Escape, Tab, Control, Shift, Caps Lock, Delete, Return, and Reset—can vary: on the Apple II and II Plus some are abbreviated or missing; on the Apple IIc all keycaps are lowercase italic; on international models, some captions are replaced by symbols (Appendix G).

The Apple IIc has two switches that the other models do not have. One switch is for changing between 40-column and 80-column display, the other is for selecting keyboard layout (Sholes versus Dvorak on USA models), or both keyboard layout and character set (on international models).

The position of the power-on light differs on the Apple II and II Plus, Apple IIe, and Apple IIc. The Apple IIc has a disk-use light as well.

---

## Character sets

The Apple II and II Plus keyboard character sets are the same. They are described in the *Apple II Reference Manual*.

The Apple IIe and Apple IIc keyboard character sets are the same: full ASCII. The standard (Sholes) layout and key assignments are described in the *Apple IIe Reference Manual*. The Dvorak layout and key assignments are described in Chapter 4 and Appendix G of this manual.

To change between the two available keyboard layouts requires modification to the main logic board on the Apple IIe, but only toggling of the keyboard switch on the Apple IIc.

Apple Computer, Inc., manufactures fully localized models (with regard to power supply and character sets) of both the Apple IIe and the Apple IIc. However, there are minor variations in keyboard layout, even among early and late production models of the same machine. For further details, refer to Appendix G of this manual or to the Apple IIe *Supplement to the Owner's Manual*.

---

---

## The speaker

The Apple IIc has two speaker features that the three previous models do not have. They are a two-channel, but monaural, audio output jack for headphones—which disconnects the internal speaker when something is plugged into it—and a volume control.

---

---

## The video display

This section discusses the general differences between Apple IIc video display capabilities and those of the other computers in the series. Note, however, that as new ROMs become available for the Apple IIe, many differences between these two machines will vanish.

---

## Character sets

The Apple II and II Plus display only uppercase characters, but they display them in three ways: normal, inverse, and flashing. The Apple IIc and IIe can display uppercase characters in all three ways, and they can display lowercase characters in the normal way. This combination is called the *primary character set*.

The Apple IIc and IIe have another character set, called the *alternate character set*, that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 5. You can switch character sets at any time by means of the AltChar soft switch, also described in Chapter 5.

Flashing display must not be used with the enhanced video firmware active. Use it in 40-column mode with the enhanced video firmware turned off; otherwise, strange displays may result, such as MouseText characters appearing in place of uppercase letters.

To be sure of compatibility with some software, you have to switch the Apple IIc keyboard to uppercase by pressing Caps Lock.

---

## MouseText

MouseText characters (Chapter 5) are available on every Apple IIc, and on the enhanced Apple IIe.

---

## Vertical blanking

A signal called *vertical blanking* indicates when a display device should stop projecting dots until the display mechanism returns from the bottom of the screen to the top to make another pass. During this interval, a program can make changes to display memory pages, and thus provide a smooth, flicker-free transition to a new display.

On the Apple IIe, vertical blanking (VBL) is a signal whose level must be polled. (VBL is not available to software on the Apple II or II Plus.) On the Apple IIc, vertical blanking is an interrupt (VBLInt) that occurs on the trailing edge of the active-low VBL signal. Programs intended to run on all Apple II series computers must take this difference into account.

---

## Display modes

All models have 40-column text mode, low-resolution graphics mode, high-resolution graphics mode, and mixed graphics and text modes. The Apple IIe (revision B motherboard) with an Apple Extended 80-Column Text Card, and the Apple IIc have double high-resolution graphics mode also.

---

---

## Disk I/O

The Apple II, II Plus, and IIe can support up to six disk drives (although four is the recommended maximum) attached in controller cards plugged into slots 6, 5, and 4. The Apple IIc supports up to two disk drives: its built-in drive (treated as slot 6, drive 1), and one external disk drive (treated as slot 6, drive 2; also treated as slot 7, drive 1 under ProDOS) for external-drive startup purposes.

---

**UniDisk 3.5** The Apple IIc with UniDisk 3.5 support does not use slot 7, drive 1 for external drives. They are handled through the Smartport described in Chapter 6. The firmware for slot 7 (\$C7xx) is needed for other parts of the firmware.

---

---

---

## Serial I/O

The Apple IIc serial ports (ports 1 and 2) are similar to Super Serial Cards installed in slots 1 and 2 of an Apple IIe. The serial port commands are a slightly modified subset of Super Serial Card commands. This subset includes all the commands supported by the earlier Apple Serial Interface Card and Communication Card.

---

### Serial ports versus serial cards

There are several important differences between Apple IIc serial ports and other Apple II series computers with serial cards installed in them.

Apple IIc serial ports have no switches. Instead, initial values are moved from firmware locations into auxiliary memory when the power is turned on. Changes made to these values in auxiliary memory remain in effect until the power is turned off. Pressing Open Apple-Control-Reset does not change them.

When the port itself is turned on (with an IN or PR command), the initial values in auxiliary memory are placed in the main memory screen holes assigned to the port. These characteristics can be changed by the port commands. The changed characteristics remain in effect until the port is turned off and then on again (with PR and IN commands).

The command syntax for the Apple IIc ports also differs from the syntax for serial cards. A separate command character, Control-A or Control-I, must precede each individual port command, whereas several commands to a serial card can be strung together between the command character and a carriage return character.

The letters used for some of the commands have been changed from those used with the Super Serial Card (such as S instead of B for sending a BREAK signal). Each serial port command letter is unique, to simplify command interpretation.

Changing the command character from Control-A to Control-I, or vice versa, makes the Super Serial Card change from communication mode to printer mode and back; this is not the case with Apple IIc serial ports. With the Apple IIc, use the *System Utilities* disk to change modes.

Super Serial Card commands support some functions that Apple IIc serial port commands don't support: translating incoming characters, such as changing lowercase to uppercase (for the benefit of the Apple II or II Plus); delaying after sending carriage return, line feed, or form feed, and so on.

---

#### UniDisk 3.5

Several new serial port commands are available on the Apple IIc with UniDisk 3.5 support. These commands have been added to make it easier to write programs that are also compatible with the Super Serial Card. See Chapters 7 and 8 for these new commands.

---

Following a Control-I nnnN command, the Apple IIc automatically generates a carriage return after nnnN characters; with the Super Serial Card, you need to turn this on with Control-I C.

---

### Serial I/O buffers

The communication port firmware uses auxiliary memory page \$08 as an input and output buffer. By doing so, the firmware can keep up with higher baud rates. It can also hide data from the Monitor, Applesoft, and other system software.

Programs written for the Apple IIe or IIc can, of course, store information in auxiliary memory page \$08. However, such information is destroyed when the communication port is activated.

---

---

## Mouse and hand controllers

The DB-9 back panel connector on the Apple IIc is used for both the mouse and hand controllers. On the Apple IIe, the DB-9 connector supports hand controllers only; the mouse must use the connector on the interface card.

---

### Mouse input

The Apple IIc provides built-in firmware support for a mouse connected to the DB-9 mouse and hand controller connector. Apple IIc mouse support includes mouse movement and button interrupts (and vertical blanking interrupts for synchronization with the display); Apple IIe mouse support relies on polling VBL instead of vertical blanking interrupts.

As a result of how interrupts are handled on the two machines, the mouse firmware routine calls function somewhat differently for the Apple IIc and Apple IIe. However, using the calls in the manner described in Chapter 9 ensures mouse support compatibility between the two machines. The ratio of mouse movement to cursor movement is different on the Apple IIc from on the Apple IIe.

---

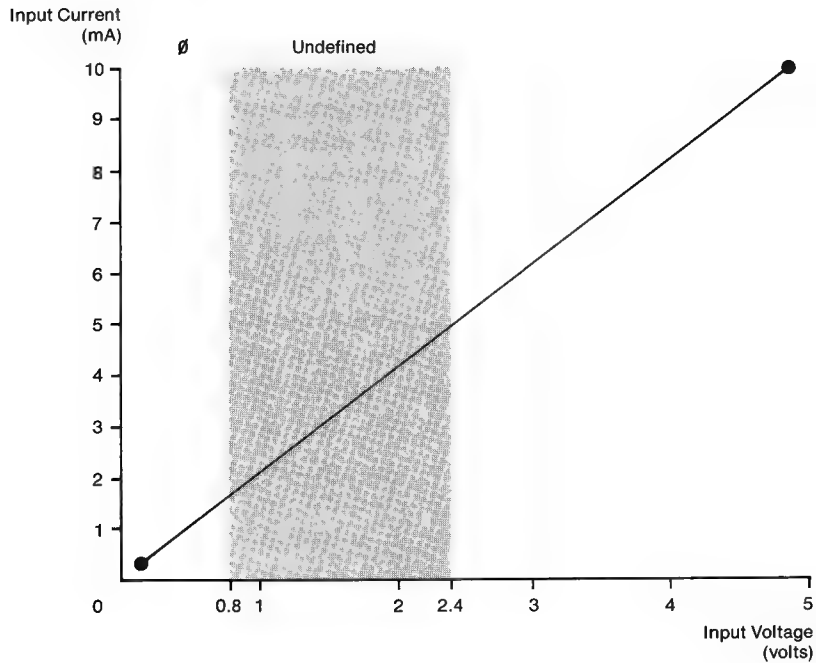
### Hand controller input and output

The Apple II, II Plus, and IIe have a 16-pin game I/O connector inside the case that supports three switch inputs, four analog (paddle) inputs, and four annunciator outputs. The Apple IIe and Apple IIc have a DB-9 back panel connector that supports the three switch inputs and two paddle inputs (plus two more on the internal GAME I/O connector of the Apple II, II Plus, and IIe).

The Apple IIc does not support the four annunciator outputs.

The characteristic response curve for hand controllers differs for the Apple IIc from that of the Apple II, II Plus, and IIe. Compare Figure F-1 with Figure 11-42. This was done so the hardware would support identifiable mouse and hand controller signals using the same circuits.

The paddle-timing circuit on the Apple II Plus is slightly different from the one on the Apple IIe and IIc. On the Apple IIe and IIc the 100-ohm fixed resistor is between the NE556 discharge lead and the capacitor; the variable resistor in the paddle is connected directly to the capacitor. On the Apple II Plus, the capacitor is connected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.



**Figure F-1**  
Apple II, II Plus, and IIe hand controller signals

---



---

## Cassette I/O

The Apple II, II Plus, and IIe all have cassette input and output jacks, memory locations, and Monitor support. The Apple IIc does not.

---

**UniDisk 3.5** If you plan to run a program on your Apple IIc that handles cassette I/O, make sure that it does not access \$C028. The Apple IIc with UniDisk 3.5 support uses address \$C028 to toggle between its two 16K banks of memory.

---

---

---

## Hardware

Besides the different microprocessors used in various models in the Apple II series, there are important differences in power specifications and custom chips.

---

### Power

The power supplies for the Apple II, II Plus, and IIe are essentially the same. The floor transformer and voltage converter for the Apple IIc have smaller capacity for current and heat dissipation. Therefore, it is important to observe the load limits specified in each of the reference manuals.

---

### Custom chips

The Apple IIe custom chips (memory management unit and input/output unit) replaced dozens of Apple II Plus chips, and added the functionality of dozens more. The Apple IIc has custom MMU and IOU chips, too, but they represent different bonding options, and so their pin assignments are not compatible.

In addition, the Apple IIc has a custom general logic unit (GLU), timing generator (TMG), and disk controller unit (also known as an Integrated Woz Machine, or IWM). The Apple IIc has two hybrid units (AUD and VID) for audio and video amplification.



## Appendix G



# USA and International Models

This appendix repeats some of the keyboard information given in Chapter 4 for the two USA keyboard layouts, for easy comparison with the other layouts available. Following these is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

---

---

### Keyboard layouts and codes

Each of the following subsections has a keyboard illustration and a table of the codes that result from the possible keystrokes. Note, however, that Table G-1 is the basic table of keystrokes and their codes. For simplicity, subsequent tables (up to Table G-7) list only the keystrokes and codes that differ from those in Table G-1.

For example, pressing the A key produces *a* (hexadecimal 61); pressing Shift-A produces uppercase *A* (hexadecimal 41); pressing Control-A or Control-Shift-A produces *SOH* (the ASCII Start Of Header control character, hexadecimal 01). You can tell that this key has the same effect on all keyboards because nothing appears in Tables G-2 through G-7 for that key.

A quick way to find out which characters in the ASCII set change on international keyboards is to check Table G-8. In fact, only a few of them change. The pairing of characters on keys varies more.

- ❖ *Note:* On all but the French and Italian keyboards, Caps Lock affects only keys that can produce both lowercase letters (with or without an accent) and their uppercase equivalents. With these keys, Caps Lock down is equivalent to holding down Shift, resulting in uppercase instead of lowercase. If a key produces only a lowercase version of an accented letter, then Caps Lock does not affect it.

On the French and Italian keyboards, Caps Lock shifts all the keys. Furthermore, on the French keyboard, when Caps Lock is down the Shift key undoes the shifting.

The shapes and arrangement of keys in Figures G-1 and G-2 follow the ANSI (American National Standards Institute) standard, which is used mainly in North and South America. The shapes and arrangement of keys in Figure G-3 follow the ISO (International Standards Organization) standard used in Europe and elsewhere.

The only differences between the ANSI and ISO versions of the USA keyboard are

- the shapes of three keys: the left Shift key, Caps Lock, and Return
- the resulting repositioning of two keys (| and ~) in Figures G-1 and G-3
- for some countries, the arrow symbols on Tab, Caps Lock, Return, and the two Shift keys (as shown in Figure G-3)

---

## USA standard (Sholes) keyboard

Figure G-1 shows the standard (Sholes) keyboard as it is laid out for USA models of the Apple IIc with the keyboard switch up. Table G-1 lists the ASCII codes resulting from all simple and combination keystrokes on this keyboard.



**Figure G-1**  
USA standard (or Sholes) keyboard, keyboard switch up

**Table G-1**  
Keys and ASCII codes

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
Delete	7F	DEL	7F	DEL	7F	DEL	7F	DEL
Left Arrow	08	BS	08	BS	08	BS	08	BS
Tab	09	HT	09	HT	09	HT	09	HT
Down Arrow	0A	LF	0A	LF	0A	LF	0A	LF
Up Arrow	0B	VT	0B	VT	0B	VT	0B	VT
Return	0D	CR	0D	CR	0D	CR	0D	CR
Right Arrow	15	NAK	15	NAK	15	NAK	15	NAK
Escape	1B	ESC	1B	ESC	1B	ESC	1B	ESC
Space	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C	,	2C	,	3C	<	3C	<
- _	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0 )	30	0	30	0	29	)	29	)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#

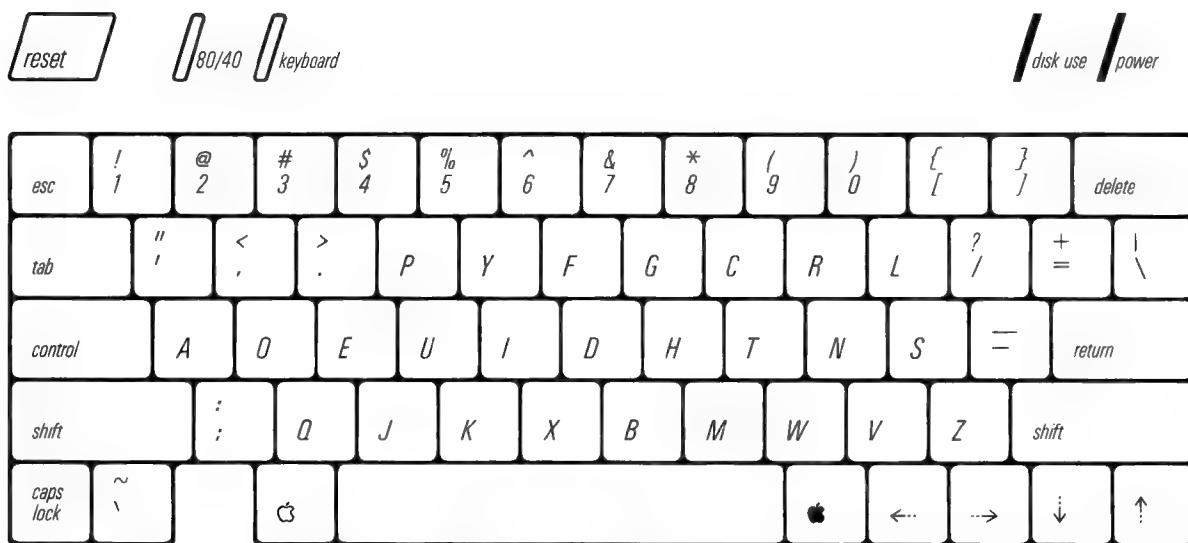
**Table G-1** (continued)  
Keys and ASCII codes

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (	39	9	39	9	28	(	28	(
; :	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[ {	5B	[	1B	ESC	7B	{	1B	ESC
\	5C	\	1C	FS	7C		1C	FS
] }	5D	]	1D	GS	7D	}	1D	GS
! ~	60	!	60	!	7E	~	7E	-
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

### USA simplified (Dvorak) keyboard

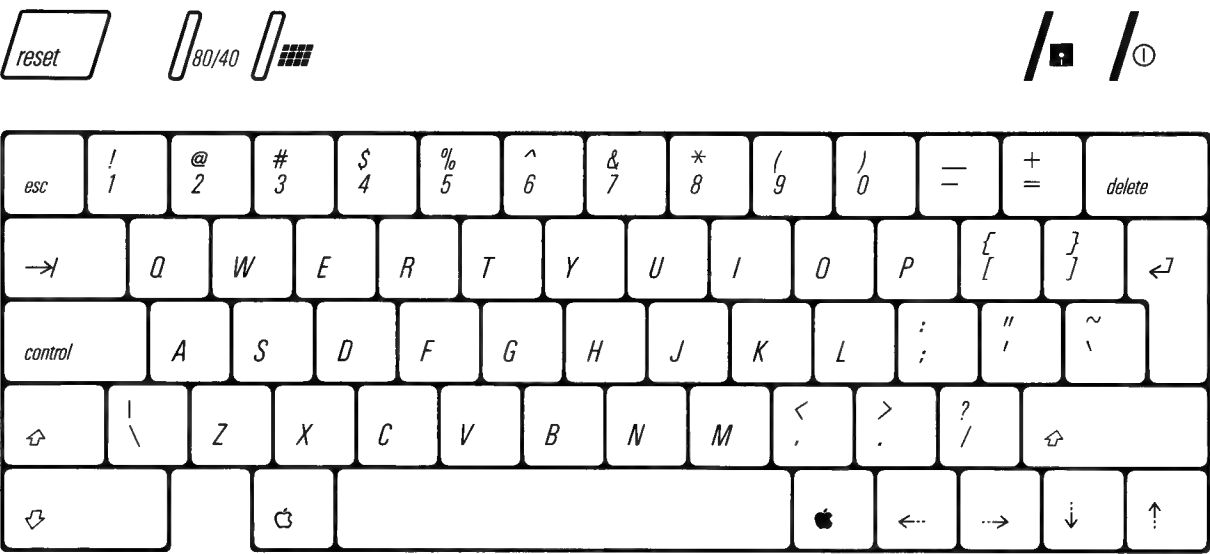
Figure G-2 shows the Dvorak layout of the USA keyboard. Characters are paired up on keys in exactly the same way as on the USA standard keyboard; only individual key positions are changed. In fact, you can change the keycap arrangement to match Figure G-2, lock the keyboard switch in its down position, and have a working Dvorak keyboard. All keystrokes produce the same ASCII codes as those shown in Table G-1.



**Figure G-2**  
USA simplified (or Dvorak) keyboard, keyboard switch down

## ISO layout of USA keyboard

Figure G-3 shows the layout of all ISO European keyboards (except the Italian keyboard) when the keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table G-1.



**Figure G-3**  
ISO version of USA standard keyboard, keyboard switch up

## English keyboard

With the keyboard switch up, the English model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the English model keyboard layout is as shown in Figure G-4. The change in ASCII code production (from that in Table G-1) is shown in Table G-2.

The only changed character is the substitution of the British pound-sterling symbol (£) for the cross-hatch symbol (#) on the shifted 3-key.



**Figure G-4**  
English keyboard, keyboard switch down

**Table G-2**  
English keyboard code differences from Table G-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
3 £	33	3	33	3	23	£	23	£

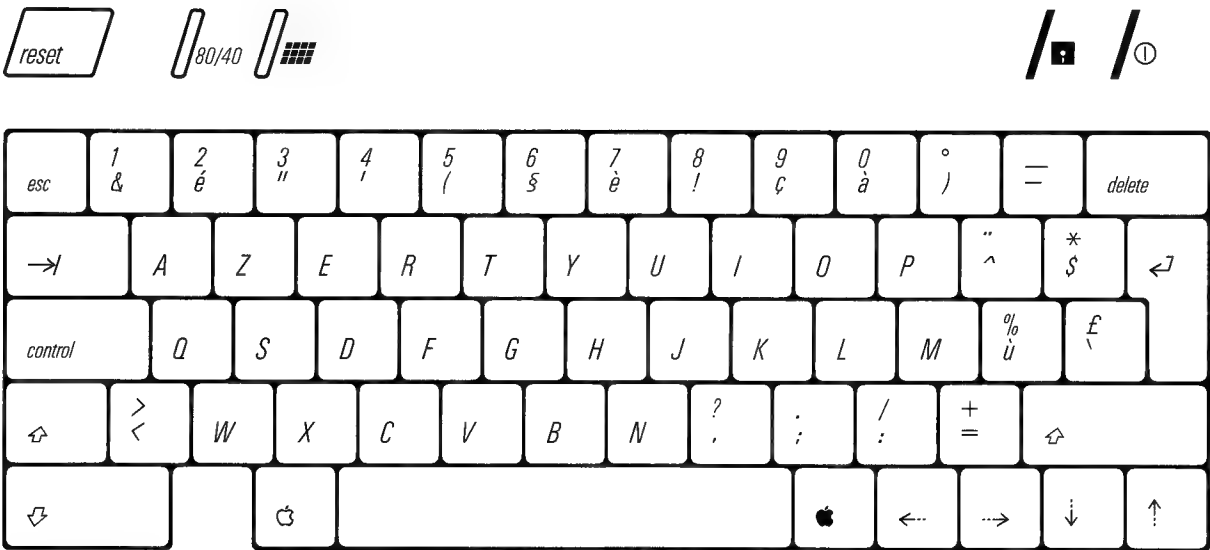
*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

# French keyboard

With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the French model keyboard layout is as shown in Figure G-5. The changes in ASCII code production (from that in Table G-1) are shown in Table G-3.

Note that on the French keyboard, Caps Lock shifts to the upper characters on all keys. With Caps Lock on, Shift “unshifts” to the lower character on any key pressed with it.



**Figure G-5**  
French keyboard, keyboard switch down

**Table G-3**  
French keyboard code differences from Table G-1

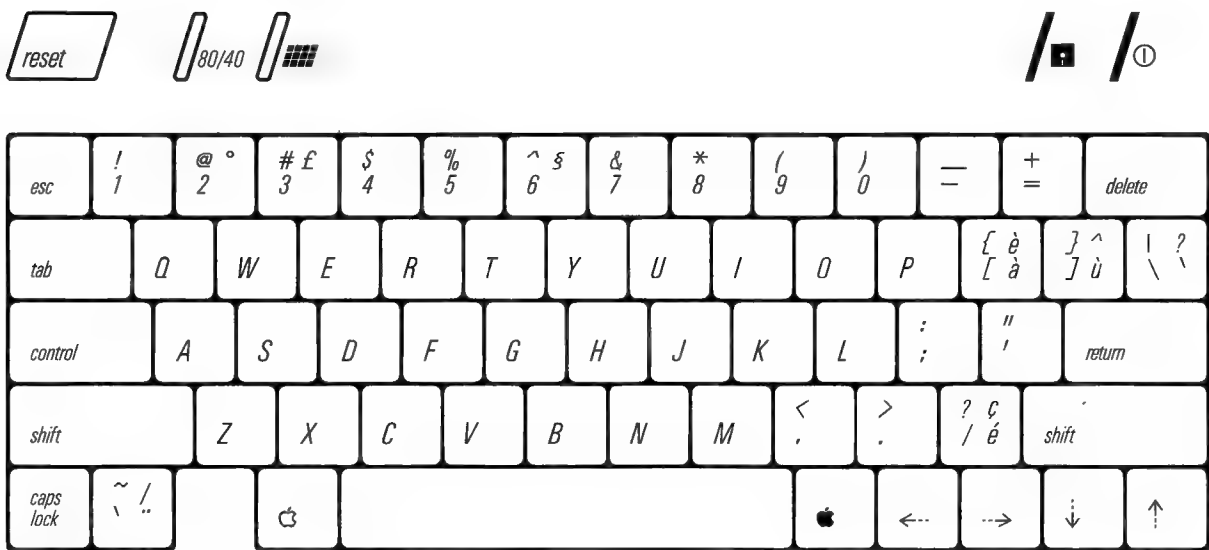
Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
& 1	26	&	26	&	31	1	31	1
é 2	7B	é	7B	é	32	2	32	2
" 3	22	"	22	"	33	3	33	3
' 4	27	'	27	'	34	4	34	4
( 5	28	(	28	(	35	5	35	5
§ 6	5D	§	1D	GS	36	6	1D	GS
è 7	7D	è	7D	è	37	7	37	7
! 8	21	!	21	!	38	8	38	8
ç 9	5C	ç	1C	FS	39	9	1C	FS
à 0	40	à	00	NUL	30	0	00	NUL
) °	29	)	1B	ESC	5B	°	1B	ESC
^ "	5E	^	1E	RS	7E	"	1E	RS
\$ *	24	\$	24	\$	2A	*	2A	*
ù %	7C	ù	7C	ù	25	%	25	%
` £	60	`	60	`	23	£	23	£
< >	3C	<	3C		3E	>	3E	>
, ?	2C	,	2C	,	3F	?	3F	?
; .	3B	;	3B	;	2E	.	2E	.
: /	3A	:	3A	:	2F	/	2F	/

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

## Canadian keyboard

With the keyboard switch up, the Canadian model of the Apple IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Canadian model keyboard layout is as shown in Figure G-6. The changes in ASCII code production (from that in Table G-1) are shown in Table G-4.



**Figure G-6**  
Canadian keyboard, keyboard switch down

**Table G-4**  
Canadian keyboard code differences from Table G-1

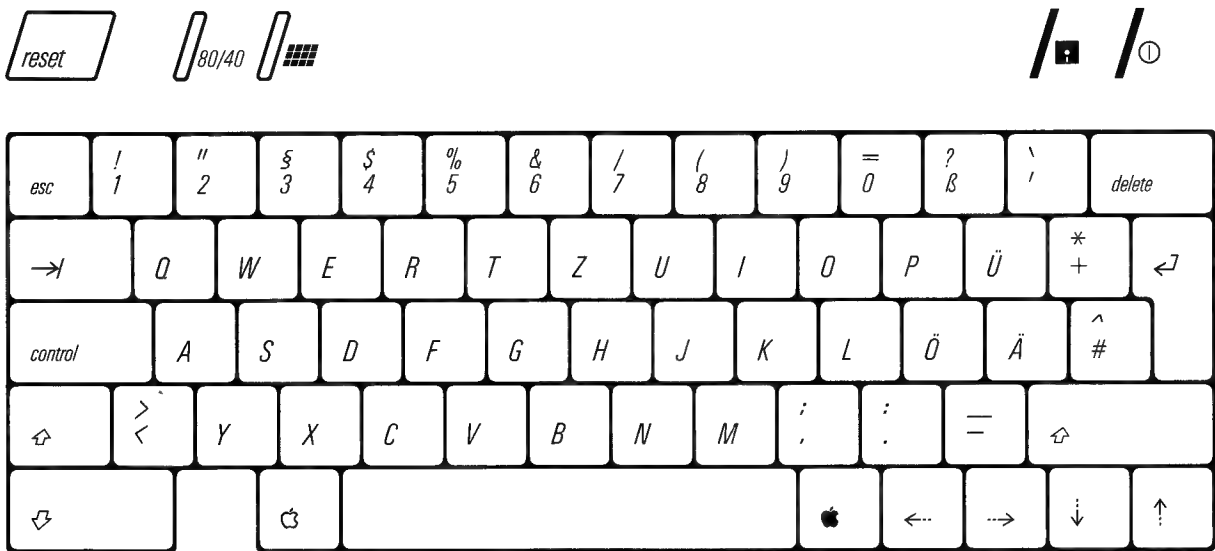
Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
2 °	32	2	00	NUL	5B	°	00	NUL
3 £	33	3	33	3	23	£	23	£
6 §	36	6	RS	1E	5D	§	RS	1E
à è	40	à	7F	DEL	7D	è	7F	DEL
ù ^	7C	ù	7C	ù	5E	^	5E	^
` ?	60	`	ESC	1B	3F	?	1D	GS
é ç	7B	é	1C	FS	5C	ç	1C	FS
" /	7E	"	7E	"	2F	/	2F	/

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

# German keyboard

With the keyboard switch up, the German model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the German model keyboard layout is as shown in Figure G-7. The change in ASCII code production (from that in Table G-1) is shown in Table G-5.



**Figure G-7**  
German keyboard, keyboard switch down

**Table G-5**

German keyboard code differences from Table G-1

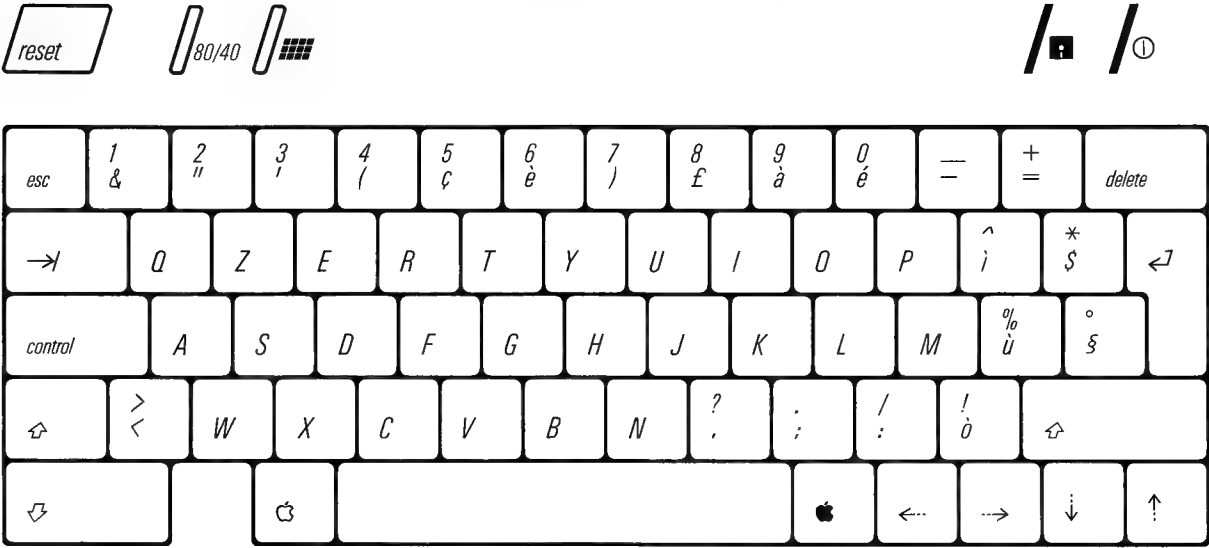
Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
2 "	32	2	32	2	22	"	22	"
3 §	33	3	00	NUL	40	§	00	NUL
6 &	36	6	36	6	26	&	26	&
7 /	37	7	37	7	2F	/	2F	/
8 (	38	8	38	8	28	(	28	(
9 )	39	9	39	9	29	)	29	)
0 =	30	0	30	0	3D	=	3D	=
ß ?	7E	ß	7E	ß	3F	?	3F	?
Ü	7D	Ü	1D	GS	5D	Ü	1D	GS
+ *	2B	+	2B	+	2A	*	2A	*
Ö	7C	Ö	1C	FS	5C	Ö	1C	FS
Ä	7B	Ä	1B	ESC	5B	Ä	1B	ESC
# ^	23	#	1E	RS	5E	^	1E	RS
< >	3C	<	3C	<	3E	>	3E	>
, ;	2C	,	2C	,	3B	;	3B	;
. :	2E	.	2E	.	3A	:	3A	:

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

## Italian keyboard

With the keyboard switch down, the Italian model keyboard layout is as shown in Figure G-8. The change in ASCII code production (from that in Table G-1) is shown in Table G-6.

With the keyboard switch up, the Italian model keyboard produces exactly the same ASCII codes for each key, but what is displayed differs for the ten characters indicated with the circled numbers 0, 2–5, and 7–11 in Table G-8.



**Figure G-8**  
Italian keyboard, keyboard switch down

**Table G-6**

Italian keyboard code differences from Table G-1

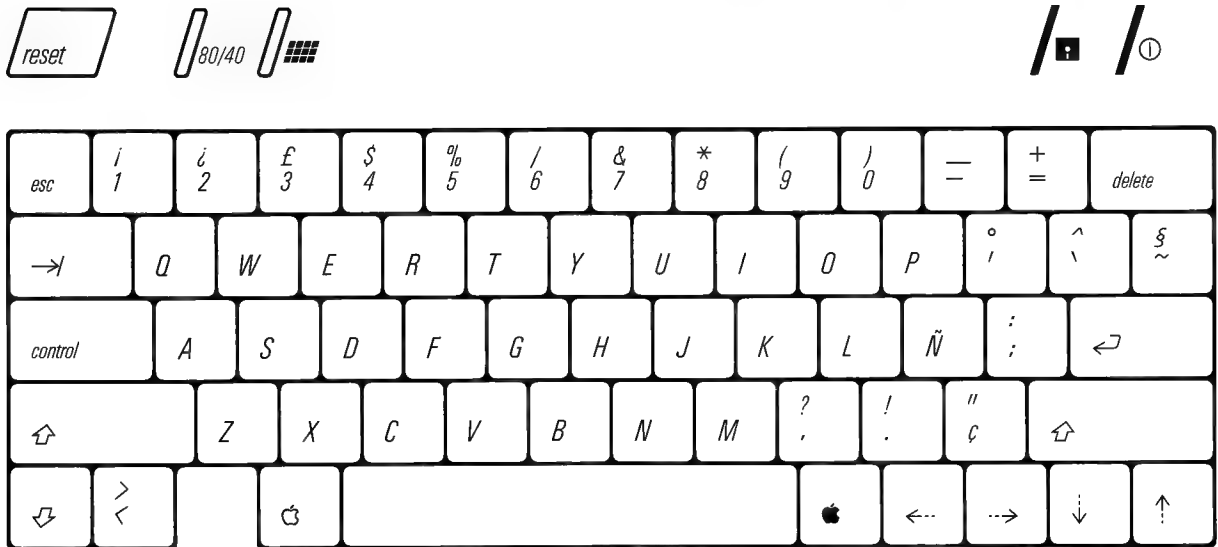
Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
& 1	26	&	26	&	31	1	31	1
" 2	22	"	22	"	32	2	32	2
' 3	27	'	27	'	33	3	33	3
( 4	28	(	28	(	34	4	34	4
ç 5	5C	ç	1C	FS	35	5	1C	FS
è 6	7D	è	7D	è	36	6	36	6
) 7	29	)	29	)	37	7	37	7
£ 8	23	£	23	£	38	8	38	8
à 9	7B	à	7B	à	39	9	39	9
é 0	5D	é	1D	GS	30	0	1D	GS
ì ^	7E	ì	1E	RS	5E	^	1E	RS
\$ *	24	\$	24	\$	2A	*	2A	▪
ù %	60	ù	60	ù	25	%	25	%
§ °	40	§	00	NUL	5B	°	1B	ESC
< >	3C	<	3C	<	3E	>	3E	>
, ?	2C	,	2C	,	3F	?	3F	?
; .	3B	;	3B	;	2E	.	2E	.
: /	3A	:	3A	:	2F	/	2F	/
ò !	7C	ò	7C	ò	21	!	21	!

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

## Western Spanish keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the Apple IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Western Spanish model keyboard layout is as shown in Figure G-9. The change in ASCII code production (from that in Table G-1) is shown in Table G-7.



**Figure G-9**  
Western Spanish keyboard, keyboard switch down

**Table G-7**

Western Spanish keyboard code differences from Table G-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
1 ¡	31	1	31	1	5B	¡	5B	¡
2 ¢	32	2	32	2	5D	¢	5D	¢
3 £	33	3	33	3	23	£	23	
6 /	36	6	36	6	2F	/	2F	/
˘ °	27	˘	27	˘	7B	°	7B	°
˘ ^	60	˘	00	NUL	5E	^	00	NUL
˘ §	7E	˘	7F	DEL	40	§	7F	DEL
Ñ	7C	Ñ	1C	FS	5C	Ñ	1C	FS
, ?	2C	,	2C	,	3F	?	3F	?
. !	2E	.	2E	.	21	!	21	!
§ "	7D	§	1D	GS	22	"	1D	GS
< >	3C	<	1E	RS	3E	>	1E	RS

*Note:* Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

## ASCII character sets

Table G-8 lists the ASCII (American National Standard Code for Information Interchange) codes that the Apple IIc uses, as well as the decimal and hexadecimal equivalents. Where there are differences between character sets, an asterisked number in the main table refers to a column in the following part of the table.

**Table G-8**

ASCII code equivalents

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
NUL	00	00	SP	32	20	2*	64	40	7*	96	60
SOH	01	01	!	33	21	A	65	41	a	97	61
STX	02	02	"	34	22	B	66	42	b	98	62
ETX	03	03	0*	35	23	C	67	43	c	99	63
EOT	04	04	1*	36	24	D	68	44	d	100	64
ENQ	05	05	%	37	25	E	69	45	e	101	65
ACK	06	06	&	38	26	F	70	46	f	102	66
BEL	07	07	'	39	27	G	71	47	g	103	67

**Table G-8 (continued)**  
ASCII code equivalents

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
BS	08	08	(	40	28	H	72	48	h	104	68
HT	09	09	)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	8	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	3*	91	5B	8*	123	7B
FS	28	1C	<	60	3C	4*	92	5C	9*	124	7C
GS	29	1D	=	61	3D	5*	93	5D	10*	125	7D
RS	30	1E	>	62	3E	6*	94	5E	11*	126	7E
US	31	1F	?	63	3F	-	95	5F	DEL	127	7F

The following characters correspond to those followed by an asterisk in the preceding part of the table.

*	0	1	2	3	4	5	6	7	8	9	10	11
Hexadecimal	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
English (USA)	#	\$	@	[	\	]	^	`	{		}	~
English (UK)	£	\$	@	[	\	]	^	`	{		}	~
German	#	\$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
French	£	\$	à	°	ç	§	^	`	é	Û	è	..
Italian	£	\$	§	°	ç	é	^	Ù	à	ò	è	Ì
Spanish	£	\$	§	í	Ñ	¿	^	`	°	ñ	ç	~

---

---

## Certification

In the countries where it is applicable, the following product safety certification supplements the USA FCC Class B notice printed on the inside front cover of this manual. The safety instructions apply to all countries.

---

## Product safety

This product is designed to meet the requirements of safety standard IEC 380, Safety of Electrically Energized Office Machines.

---

## Important safety instructions

This equipment is intended to be electrically grounded. This product is equipped with a plug having a third (grounding) pin. This plug will fit only into a grounding-type alternating current outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

---

---

## Power supply specifications

The basic specifications of the power supply furnished with the Apple IIc for use in Europe and other countries having 50-Hz alternating current are shown in Table G-8.

**Table G-8**  
50-Hz power supply specifications

<b>Line voltage</b>	199 to 255 VAC, 50 Hz
<b>Maximum input power consumption</b>	25 W
<b>Supply voltage</b>	+15 VDC (nominal)
<b>Supply current</b>	1.2 A (nominal)



# Appendix H



## Conversion Tables

This briefly discusses bits and bytes and what they can represent, and peripheral identification numbers. It also contains conversion tables for hexadecimal to decimal and negative decimal, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

---

---

### Bits and bytes

This section discusses the relationships between bit values and their position within a byte. Here are some rules of thumb regarding the 65C02:

- A **bit** is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIc are listed in Table H-1.
- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits make a **nibble** (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only 10 of the 16 digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure H-1).

- One **byte** can represent any of  $16 \times 16$  (or 256) values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- One memory position in the Apple IIc contains one 8-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables H-6 through H-9 list some of the ways bytes are commonly interpreted.
- Two bytes make a **word**. The 16 bits of a word can represent any one of  $256 \times 256$  (or 65,536) different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65,536 (64K) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

**Table H-1**  
What a bit can represent

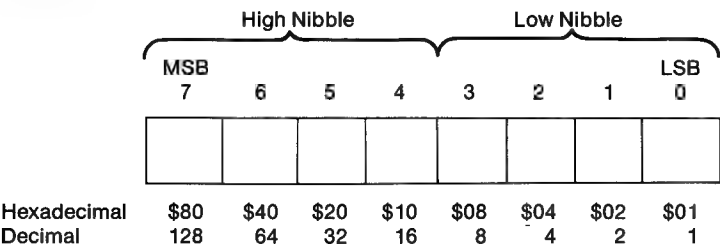
Context	Representing	0 =	1 =
Binary number	Place value	0	$1 \times$ that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear*	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier

**Table H-1 (continued)**  
 What a bit can represent

Context	Representing	0 =	1 =
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

\* Sometimes ambiguously termed *reset*.

**Figure H-1**  
 Bits, nibbles, and bytes



**Table H-2**  
 Values represented by a nibble

Binary	Hex	Dec	Binary	Hex	Dec
0000	\$0	0	1000	\$8	8
0001	\$1	1	1001	\$9	9
0010	\$2	2	1010	\$A	10
0011	\$3	3	1011	\$B	11
0100	\$4	4	1100	\$C	12
0101	\$5	5	1101	\$D	13
0110	\$6	6	1110	\$E	14
0111	\$7	7	1111	\$F	15

---

---

## Hexadecimal and decimal

Use Table H-3 for conversion of hexadecimal and decimal numbers.

**Table H-3**  
Hexadecimal/decimal conversion

Digit	\$x000	\$0x00	\$00x0	\$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

For example:

\$3C = ?	\$FD47 = ?
\$30 = 48	\$F000 = 61440
\$0C = 12	\$ D00 = 3328
	\$ 40 = 64
\$3C = 60	\$ 7 = 7
	\$FD47 = 64839

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have 0 left. Add up the hexadecimal numbers.

16215	= \$ ?		
16215	- 12288	= 3927	12288 = \$7000
3927	- 3840	= 87	3840 = \$ F00
87	- 80	= 7	80 = \$ 50
	7		7 = \$ 7
			<hr/>
			16215 = \$7F57

If a number is larger than decimal 32,767, Applesoft BASIC allows and Integer BASIC requires you to use the negative-decimal equivalent of the number. Table H-4 is set up to make it easy for you to convert a hexadecimal number directly to a negative-decimal number.

Digit	\$x000	\$\$0x00	\$\$\$00x0	\$\$\$\$000x
F	0	0	0	-1
E	-4096	-256	-16	-2
D	-8192	-512	-32	-3
C	-12288	-768	-48	-4
B	-16384	-1024	-64	-5
A	-20480	-1280	-80	-6
9	-24576	-1536	-96	-7
8	-28672	-1792	-112	-8
7		-2048	-128	-9
6		-2304	-144	-10
5		-2560	-160	-11
4		-2816	-176	-12
3		-3072	-192	-13
2		-3328	-208	-14
1		-3584	-224	-15
0		-3840	-240	-16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0's included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative-decimal number.

For example:

\$C010	=	- ?
\$C000:		-12288
\$ 000:		- 3840
\$ 10:		- 224
\$ 0:		- 16
<hr/>		
\$C010		-16368

To convert a negative-decimal number directly to a positive-decimal number, add it to 65,536. (This addition ends up looking like subtraction.)

For example:

-151 = + ?  
 65536 + (-151) = 65536 - 151 = 65385

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive-decimal number, then use Table H-3.

---

---

## Peripheral identification numbers

Many Apple products now use peripheral identification numbers (called *PIN numbers*) as shorthand to designate serial device characteristics. The Apple II series *Universal Utilities* disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table H-5 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

For example:

252/1111 means:

Communication mode.	Do not echo output to display.
8 data bits, 1 stop bit.	No line feed after carriage return.
300 baud (bits per second).	Do not generate carriage returns.
No parity.	

**Table H-5**  
PIN numbers

		x	x	x	/	x	x	x	x
1 = Printer mode									
2 = Communication mode*									
1 = 6 data bits, 1 stop bit									
2 = 6 data bits, 2 stop bits									
3 = 7 data bits, 1 stop bit									
4 = 7 data bits, 2 stop bits									
5 = 8 data bits, 1 stop bit									
6 = 8 data bits, 2 stop bits									
1 = 110 bits per second									
2 = 300 bits per second									
3 = 1200 bits per second									
4 = 2400 bits per second									
5 = 4800 bits per second									
6 = 9600 bits per second									
7 = 19200 bits per second									
1 = No parity									
2 = Even parity (total on = even)									
3 = Odd parity (total on = odd)									
4 = MARK parity (parity bit = 1)									
5 = SPACE parity (parity bit = 0)									
1 = Do not echo output on screen									
2 = Echo output on screen									
1 = Do not generate LF after CR									
2 = Generate LF after CR									
1 = Do not generate CR*									
2 = Generate CR after 40 characters									
3 = Generate CR after 72 characters									
4 = Generate CR after 80 characters									
5 = Generate CR after 132 characters									

\* If you select communication mode, then seventh digit must equal 1. This value is supplied automatically when you use the UUD.

---

---

## Eight-bit code conversions

Tables H-6 through H-9 show the entire ASCII character set. Note that character values are shown with the high bit off. Unless otherwise noted, all ASCII character values above \$7F (127 decimal) generate the same character as that value with the high bit off. Here is how to interpret these tables:

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *H*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *H*.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity *H*, 01001000 for an even-parity *H*.
- The *ASCII Char* column gives the ASCII character name.
- The *Interpretation* column spells out the meaning of special symbols and abbreviations, where necessary.
- The *What to type* column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked *Pri* and *Alt* indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

Note that the values \$40 through \$5F (and \$C0 through \$DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so, or if the firmware is bypassed.

- ❖ *Note:* The primary and alternate displayed character sets in Tables H-6 through H-9 are the result of firmware mapping. The character generator ROM actually contains only one character set. The firmware mapping procedure is described in Chapter 3.

**Table H-6**  
Control characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
0000000	0	\$00	NUL	Blank (null)	Control-@	@	@
0000001	1	\$01	SOH	Start of header	Control-A	A	A
0000010	2	\$02	STX	Start of text	Control-B	B	B
0000011	3	\$03	ETX	End of text	Control-C	C	C
0000100	4	\$04	EOT	End of transm.	Control-D	D	D
0000101	5	\$05	ENQ	Enquiry	Control-E	E	E
0000110	6	\$06	ACK	Acknowledge	Control-F	F	F
0000111	7	\$07	BEL	Bell	Control-G	G	G
0001000	8	\$08	BS	Backspace	Control-H or Left Arrow-H	H	H
0001001	9	\$09	HT	Horizontal tab	Control-I or Tab	I	I
0001010	10	\$0A	LF	Line feed	Control-J or Down Arrow-J	J	J
0001011	11	\$0B	VT	Vertical tab	Control-K or Up Arrow	K	K
0001100	12	\$0C	FF	Form feed	Control-L	L	L
0001101	13	\$0D	CR	Carriage return	Control-M or Return	M	M
0001110	14	\$0E	SO	Shift out	Control-N	N	N
0001111	15	\$0F	SI	Shift in	Control-O	O	O
0010000	16	\$10	DLE	Data link escape	Control-P	P	P
0010001	17	\$11	DC1	Device control 1	Control-Q	Q	Q
0010010	18	\$12	DC2	Device control 2	Control-R	R	R
0010011	19	\$13	DC3	Device control 3	Control-S	S	S
0010100	20	\$14	DC4	Device control 4	Control-T	T	T
0010101	21	\$15	NAK	Neg. acknowledge	Control-U or Right Arrow	U	U
0010110	22	\$16	SYN	Synchronization	Control-V	V	V
0010111	23	\$17	ETB	End of text blk.	Control-W	W	W
0011000	24	\$18	CAN	Cancel	Control-X	X	X
0011001	25	\$19	EM	End of medium	Control-Y	Y	Y
0011010	26	\$1A	SUB	Substitute	Control-Z	Z	Z
0011011	27	\$1B	ESC	Escape	Control-[ or Escape	[	[
0011100	28	\$1C	FS	File separator	Control-\	\	\
0011101	29	\$1D	GS	Group separator	Control-]	]	]
0011110	30	\$1E	RS	Record separator	Control-^	^	^
0011111	31	\$1F	US	Unit separator	Control-_	_	_

**Table H-7**  
Special characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
0100000	32	\$20	SP	Space	Space bar		
0100001	33	\$21	!			!	!
0100010	34	\$22	"			"	"
0100011	35	\$23	#			#	#
0100100	36	\$24	\$			\$	\$
0100101	37	\$25	%			%	%
0100110	38	\$26	&			&	&
0100111	39	\$27	'	Apostrophe		'	'
0101000	40	\$28	(			(	(
0101001	41	\$29	)			)	)
0101010	42	\$2A	*			*	*
0101011	43	\$2B	+			+	+
0101100	44	\$2C	,	Comma		,	,
0101101	45	\$2D	-	Hyphen		-	-
0101110	46	\$2E	.	Period		.	.
0101111	47	\$2F	/			/	/
0110000	48	\$30	0			0	0
0110001	49	\$31	1			1	1
0110010	50	\$32	2			2	2
0110011	51	\$33	3			3	3
0110100	52	\$34	4			4	4
0110101	53	\$35	5			5	5
0110110	54	\$36	6			6	6
0110111	55	\$37	7			7	7
0111000	56	\$38	8			8	8
0111001	57	\$39	9			9	9
0111010	58	\$3A	:			:	:
0111011	59	\$3B	;			;	;
0111100	60	\$3C	<			<	<
0111101	61	\$3D	=			=	=
0111110	62	\$3E	>			>	>
0111111	63	\$3F	?			?	?

**Table H-8**  
Uppercase characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
1000000	64	\$40	@			@	Apple
1000001	65	\$41	A			A	Apple with key
1000010	66	\$42	B			B	Apple with key
1000011	67	\$43	C			C	Apple with key
1000100	68	\$44	D			D	Apple with key
1000101	69	\$45	E			E	Apple with key
1000110	70	\$46	F			F	Apple with key
1000111	71	\$47	G			G	Apple with key
1001000	72	\$48	H			H	Apple with key
1001001	73	\$49	I			I	Apple with key
1001010	74	\$4A	J			J	Apple with key
1001011	75	\$4B	K			K	Apple with key
1001100	76	\$4C	L			L	Apple with key
1001101	77	\$4D	M			M	Apple with key
1001110	78	\$4E	N			N	Apple with key
1001111	79	\$4F	O			O	Apple with key
1010000	80	\$50	P			P	Apple with key
1010001	81	\$51	Q			Q	Apple with key
1010010	82	\$52	R			R	Apple with key
1010011	83	\$53	S			S	Apple with key
1010100	84	\$54	T			T	Apple with key
1010101	85	\$55	U			U	Apple with key
1010110	86	\$56	V			V	Apple with key
1010111	87	\$57	W			W	Apple with key
1011000	88	\$58	X			X	Apple with key
1011001	89	\$59	Y			Y	Apple with key
1011010	90	\$5A	Z			Z	Apple with key
1011011	91	\$5B	[		Opening bracket	/	Apple with key
1011100	92	\$5C	\		Reverse slant	\	Apple with key
1011101	93	\$5D	]		Closing bracket	/	Apple with key
1011110	94	\$5E	^		Caret	^	Apple with key
1011111	95	\$5F	_		Underline	-	Apple with key

\* If the high bit is set, the MouseText characters are replaced with their equivalent in the primary character set with that value.

**Table H-9**  
Lowercase characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
1100000	96	\$60	`	Grave accent			`
1100001	97	\$61	a		/		a
1100010	98	\$62	b		"		b
1100011	99	\$63	c		#		c
1100100	100	\$64	d		\$		d
1100101	101	\$65	e		%		e
1100110	102	\$66	f		&		f
1100111	103	\$67	g		'		g
1101000	104	\$68	h		(		h
1101001	105	\$69	i		)		i
1101010	106	\$6A	j		"		j
1101011	107	\$6B	k		+		k
1101100	108	\$6C	l		,		l
1101101	109	\$6D	m		-		m
1101110	110	\$6E	n		.		n
1101111	111	\$6F	o		/		o
1110000	112	\$70	p		0		p
1110001	113	\$71	q		1		q
1110010	114	\$72	r		2		r
1110011	115	\$73	s		3		s
1110100	116	\$74	t		4		t
1110101	117	\$75	u		5		u
1110110	118	\$76	v		6		v
1110111	119	\$77	w		7		w
1111000	120	\$78	x		8		x
1111001	121	\$79	y		9		y
1111010	122	\$7A	z		:		z
1111011	123	\$7B	{	Opening brace	;		{
1111100	124	\$7C		Vertical line	<		
1111101	125	\$7D	}	Closing brace	=		}
1111110	126	\$7E	~	Overline (tilde)	>		~
1111111	127	\$7F	DEL	Delete/rubout	?		DEL



## Appendix I



# Firmware Listings

Appendix I is a listing of the source code for the Monitor firmware (including the Smartport) contained in the memory expansion version of the Apple IIc.

If you are developing products for an earlier version of the IIc, you can obtain a complete set of firmware listings from the Apple Programmer's and Developer's Association (APDA). You can also order additional technical information on other Apple products from the APDA. To obtain the listing set or other technical information, write to

Apple Programmer's and Developer's Association  
290 SW 43d Street  
Renton, WA 98055  
206-251-6548

**Table I-1**  
Main side ROM map

---

C100	serial port
C200	communications port
C300	80-column routines
C400	memory expansion card: boot code/entry point for ProDOS/entry point for DOS
C500–C58D	UniDisk 3.5 routines
C58E	miscellaneous
C600	Disk II routines
C700–C77F	Mouse entry points
C780	ROM switch routines
C7FC	last screen hole
C800	Interrupt routines
C900	Paddle patch; Mini-Assembler
CA00	Mini-Assembler; step and trace
CB00	scroll routines
CC00	pasinvert; picky; showcur; update; escape
CD00	video routines
CE00	video routines
CF00	video routines; miscellaneous
D000–F7FF	Applesoft BASIC
F800–FFFF	RESET vectors

**Table I-2**  
Auxiliary side ROM map

---

C100	Mouse interrupt handler; ACIA interrupt handler
C200	continue; ACIA routines
C300	ACIA routines; moveaux; xfer; memory test
C400	continue; switch test
C500–C57F	diagnostics
C780	ROM switch routines
C800–C87F	miscellaneous
C880–CFFF	UniDisk 3.5 routines
D000	command processor for serial and communications ports
D100	continue
D200–D249	Mouse BASIC routines
D24A–D3FF	empty
D400	basilin; hex-to-dec
D4A9–D52B	zero page test
D500–D52B	miscellaneous
D52C–D5FF	empty
D600–D700	Mouse routines
D800	execute; xdiag; xstatus; pstat0; pcn41; pstatus; xread; xwrite; prdblk; pwrblk; pread
D900	pread continue; pwrite
DA00	dospatch; dosconv; stattbl; parmtbl; undtbl; testsize; format
DB00	fmpas; fmdos; makecat; cattbl; procut
DC00	doscut; pascut
DCF0–DCFF	diagnostics (orged at \$DD00)
DD00	stattest; addresstest; rollovertest; addbustest; cleartest; fulltest
DE00	continue; computed; pass; fail; miscellaneous
DF00	print; messages
E000–FFF9	empty

01 FILE

```

SOURCE      EQU     1
INCLUDE FILE #02 =>NAMES
INCLUDE FILE #03 =>EQUATES
INCLUDE FILE #04 =>SERIAL
INCLUDE FILE #05 =>SER
INCLUDE FILE #06 =>COMM
INCLUDE FILE #07 =>C3SPACE
INCLUDE FILE #08 =>EQUATES2
INCLUDE FILE #09 =>SLINKY
INCLUDE FILE #10 =>MISC
INCLUDE FILE #11 =>BOOT
INCLUDE FILE #12 =>MOUSE
INCLUDE FILE #13 =>MCODE.X
INCLUDE FILE #14 =>SWITCHER
INCLUDE FILE #15 =>TROGUF
INCLUDE FILE #16 =>MINI
INCLUDE FILE #17 =>SCROLLING
INCLUDE FILE #18 =>ESCAPE
INCLUDE FILE #19 =>PASCAL
INCLUDE FILE #20 =>MOREMISC
INCLUDE FILE #21 =>AUTOST1
INCLUDE FILE #22 =>AUTOST2
INCLUDE FILE #23 =>BANK2
INCLUDE FILE #24 =>MINT
INCLUDE FILE #25 =>MUSTUFF
INCLUDE FILE #26 =>BANGER2
INCLUDE FILE #27 =>RW.SLINKY
INCLUDE FILE #28 =>MCODE.X.AUX
INCLUDE FILE #29 =>SWITCHER2
INCLUDE FILE #30 =>COMMAND
INCLUDE FILE #31 =>MBASIC
INCLUDE FILE #32 =>BANGER
INCLUDE FILE #33 =>MOUSEIN7.X
INCLUDE FILE #34 =>S.EXECUTE
INCLUDE FILE #35 =>S.MAKECAT
INCLUDE FILE #36 =>S.DINGO.SRC
INCLUDE FILE #37 =>VECTORS2

```

```

0000:      0000      2      x6502
0000:      0000:      3      1st on,vsym,asym
0000:      0000:      4      *****
0000:      0000:      5      *
0000:      0000:      6      * Firmware for the Apple //c
0000:      0000:      7      * Copyright Apple Computer Inc., 1983,1985,1986
0000:      0000:      8      * All Rights Reserved
0000:      0000:      9      *
0000:      0000:     10      * Initial release -- December 1983
0000:      0000:     11      * Written by Rich Williams, Ernie Beernink and James R Ruston
0000:      0000:     12      *
0000:      0000:     13      *****
0000:      0000:     14      *
0000:      0000:     15      * Revision 2 -- May 1985, Richard Williams
0000:      0000:     16      * rom expanded to 32K in 2 16K banks
0000:      0000:     17      * new features added:
0000:      0000:     18      * Protocol converter slot 5
0000:      0000:     19      * AppleTalk slot 7
0000:      0000:     20      * //e diagnostics
0000:      0000:     21      * Enhanced serial port commands
0000:      0000:     22      * Mini assembler
0000:      0000:     23      * Step and trace.
0000:      0000:     24      * most $F8 rom changes marked with a +
0000:      0000:     25      *
0000:      0000:     26      *****
0000:      0000:     27      *
0000:      0000:     28      * Revision 3 -- April 1986, Raymond Chiang
0000:      0000:     29      * mouse moved from slot 4 to slot 7
0000:      0000:     30      * appleTalk removed from slot 7
0000:      0000:     31      * memory card (slinky) added to slot 4
0000:      0000:     32      * boot sequence is now slot 4, then slot 6 and then slot 5
0000:      0000:     33      * $f8bf the rev byte will be 3. 1 and 2 will not be used
0000:      0000:     34      *
0000:      0000:     35      *****
0000:      0000:     36      F800      36      F800      EQU      $F800

INCLUDE FILE #02 =>NAMES
C100:      39      1st on,vsym,asym
C100:      40      include equates ;Equates for Video & Monitor ROM

```

Apple //c Video Firmware		20-OCT-86	06:41	PAGE 4
003D	60 A1H	EQW	\$3D	;Monitor temp
003E	61 A2L	EQW	\$3E	;Monitor temp
003F	62 A2H	EQW	\$3F	;Monitor temp
0040	63 A3L	EQW	\$40	;Monitor temp
0041	64 A3H	EQW	\$41	;Monitor temp
0042	65 A4L	EQW	\$42	;Monitor temp
0043	66 A4H	EQW	\$43	;Monitor temp
0044	67 A5L	EQW	\$44	;Monitor temp
0045	68 A5H	EQW	\$45	;Monitor temp
69 *				
70 *	Note: In Apple II, //e, both interrupts and BRK destroyed.			
71 *	Location \$45. Now only BRK destroys \$45 (ACC) and it			
72 *	also destroys \$44 (MUSPTR).			
73 *				
0044	74 MASTINT	EQW	\$44	;Machine state after BRK
0045	75 ACC	EQW	\$45	;Acc after BRK
0046	77 YREG	EQW	\$46	;X reg after break
0047	78 YREG	EQW	\$47	;Y reg after break
0048	79 STATUS	EQW	\$48	;P reg after break
0049	80 SPT	EQW	\$49	;SP after break
004E	81 RNOL	EQW	\$4E	;random counter low
004F	82 RNDB	EQW	\$4F	;random counter high
83 *				
84 *	Value equates			
85 *				
0006	86 GCONF8	EQW	\$06	;value of //e, lolly ID byte
0095	87 PICK	EQW	\$95	;CONTROL-U character
009B	88 ESC	EQW	\$9B	;what ESC generates
89 *				
90 *	Characters read by GETLN are placed in			
91 *	IN, terminated by a carriage return.			
92 *				
93 IN	EQW	\$0200		;input buffer for GETLN
94 *				
95 *	Page 3 vectors			
96 *				
03F0	97 BRKV	EQW	\$03F0	;vectors here after break
03F2	98 SOSTEV	EQW	\$03F2	;vector for warm start
03F4	99 PWREOP	EQW	\$03F4	;THIS MUST = EOR \$A5 OF SOSTEV+1
03F5	100 AMPEV	EQW	\$03F5	;APPLESOFT & EXIT VECTOR
03F8	101 USBRDR	EQW	\$03F8	;APPLESOFT USR function vector
03FE	102 NMI	EQW	\$03FE	;NMI vector
03FF	103 IRQOC	EQW	\$03FF	;Maskable interrupt vector
0400	104 LINE1	EQW	\$0400	;first line of text screen
0078	105 MSLGT	EQW	\$07F8	;owner of \$C8 space
106 *				
107 *	HARDWARE EQUATES			
108 *				
C008	109 T0ADR	EQW	\$C000	;for INH, PRF vector
C009	110 XED	EQW	\$C000	;>127 if keystroke
0001	111 CLR8COL	EQW	\$C000	;disable 80 column store
C001	112 SET8COL	EQW	\$C001	;enable 80 column store
C002	113 DMAINDRM	EQW	\$C002	;read from main 48K RAM
C003	114 RCDADRDM	EQW	\$C003	;read from alt. 48K RAM
C004	115 WRWADRDM	EQW	\$C004	;write to main 48K RAM
C005	116 WRCDADRDM	EQW	\$C005	;write to alt. 48K RAM
C008	117 SETSDOP	EQW	\$C008	;use main zero page/stack



```

03 EQUATES                               20-OCT-86 06:41 PAGE 7

Apple //c Video Firmware

002B 234 SLO7Z EQU $2B
C100: 003C 235 BOOTIMP EQU $3C
C100: 004F 236 BOOTDEV EQU $4F

238 *****
239 * Entry points for other modules
C100: 240 *****
C100: 241 pcnv equ $C980
C100: 242 bootfail equ $C9F5
C100: 243 pcnvrst equ $C9F8
C100: 244 atalk equ $C580
C100: 41 include serial
;Boot fails message
;Protocol converter reset
;Apple talk
;Equates for serial code

```

```

04 SERIAL                               20-OCT-86 06:41 PAGE 8

Serial & Communications equates

3 *****
4 *
5 * Apple Lolly communications driver
6 *
7 * By
8 * Rich Williams
9 * August 1983
10 * November 5 - j.r.huston
11 *
12 *****
13 *
14 * Command codes
15 *
16 * Default command char is ctrl-A (^A)
17 *
18 * *AnnB: Set baud rate to nn
19 * *AnnD: Set data format bits to nn
20 * *AI: Enable video echo
21 * *AL: Disable CRUF
22 * *AL: Enable CRUF
23 * *AnnM: Disable video echo & set printer width
24 * *AnnP: Set parity bits to nn
25 * *AQ: Quit terminal mode
26 * *AR: Reset the ACIA, IN#0 PR#0
27 * *AS: Send a 233 ms break character
28 * *AT: Enter terminal mode
29 * *AZ: Zap control commands
30 * *Ax: Set command char to ^x
31 * *AnnCR: Set printer width (CR = carriage return)
32 *
33 * New commands added in rev 1 E = enable D = Disable
34 *
35 * *AC E/D Column overflow
36 * *AL E/D Linefeed same as L & K
37 * *AM E/D Mask Incoming linefeeds
38 * *AX E/D Xon Xoff handshaking
39 * *AF E/D Find keyboard
40 *
41 *****
C100 42 serslot equ $C100
C200 43 comslot equ $C200
C100 44 msh ON
00BF 45 candcur equ '?' ;Cursor while in command mode
00DF 46 termcur equ ' ' ;Cursor while in terminal mode
C100 47 msh OFF
008A 48 lfeed equ $8A ;Linefeed
0091 49 xon equ $91 ;XON character
0093 50 xoff equ $93 ;XOFF character
03B8 51 sermode equ $3B8 ;D7=1 if in command D6=1 if terminal
C100 0438 52 astat equ $438 ;Acia status from int 4F9 & 4FA
C100 04B8 53 pwidth equ $4B8 ;Printer width 579 & 57A
C100 0538 54 extint equ $538 ;extint & typhed enable 5F9 & 5FA
C100 05F9 55 extint2 equ $5F9
C100 05FA 56 typhed equ $5FA ;Saves cursor while in command
C100 0679 57 oldcur equ $679 ;Saves cursor while in terminal mode
C100 067A 58 oldcur2 equ $67A ;Current escape character 6F9 & 6FA
C100 0638 59 eschar equ $638 ;Current escape character 6F9 & 6FA
C100 0688 60 flags equ $688 ;D7 = Video echo D6 = CRUF 779 & 77A

```

```

0738      0738      equ $738
077E      077E      equ $77E
04fC      04fC      equ $4fC
057C      057C      equ $57C
057C      057C      equ $57C
057C      057C      equ $57C
067C      067C      equ $67C
067F      067F      equ $67F
0800      0800      equ $800
06f8      06f8      equ $6f8
05FE      05FE      equ $5FE
BFF8      BFF8      equ $BFF8
BFF9      BFF9      equ $BFF9
BFFA      BFFA      equ $BFFA
BFFB      BFFB      equ $BFFB
42         42         include ser

```

```

05 SER      05 SER      Serial output port routine
C100:      C100:      3 'org serialot
C100:2C 89 C1 4 bit serrts
C103:70 0C C111 5 bvs entr1
C105:38 6 sec
C106:90 7 dfb $90
C107:18 8 clc
C108:88 9 clv
C109:50 06 C111 10 bvc entr1
C10B:01 12 dfb $01
C10C:31 13 dfb $31
C10D:9E 14 dfb >plnit
C10E:A8 15 dfb >pread
C10F:B4 16 dfb >lwrtie
C110:BB 17 dfb >pistatus
C111:DA 19 entr1 phx #<erslot
C112:A2 C1 20 ldx #<erslot
C114:4C 1C C2 21 jmp setup
C117:90 03 C11C 22 serport bcc serisout
C119:4C E3 C7 23 jmp swzzam
C11C:0A 24 serisout asl A
C11D:7A 25 ply
C11E:5A 26 phy
C11F:BD B8 04 27 lda pwidth,x
C122:F0 42 C166 28 beq prnow
C124:A5 24 29 lda ch
C126:B0 1C C144 30 bcs servid
C128:DD B8 04 31 cmp pwidth,x
C12B:90 03 C130 32 bcc chok
C12D:BD 38 07 33 lda col,x
C130:DD 38 07 34 chok cmp col,x
C133:B0 0B C140 35 bcs finch
C135:C9 11 36 cmp #311
C137:B0 11 C14A 37 bcs prnt
C139:09 F0 38 ora #3F0
C13B:3D 38 07 39 and col,x
C13E:65 24 40 adc ch
C140:85 24 41 finch sta ch
C142:80 06 C14A 42 bra prnt
C144:C5 21 43 servid cmp pwidth
C146:90 02 C14A 44 blt prnt
C148:64 24 45 stz ch

```

```

C14A:      C14A:      47 * We have a char to print
C14A:7A 48 prnt ply
C14B:5A 49 phy
C14C:BD 38 07 50 lda col,x
C14F:DD B8 04 51 cmp pwidth,x
C152:B0 08 C15C 52 bge toofar
C154:C5 24 53 cmp ch
C156:B0 0E C166 54 bge prnow
C158:A9 40 55 lda #540
C15A:80 02 C15E 56 bra tab
C15C:A9 1A 57 toofar lda #51A
C15E:C0 80 58 tab cpy #580

```

```

;Set V to indicate initial entry
;Always taken
;Input entry point
;BCC opcode
;V = 0 since not initial entry
;Always taken
;pascal signature byte
;device signature
;Save the reg
;X = Ch
;Set mslot, etc
;Only output allowed
;Reset the hooks
;A = flags
;Get char
;Formatting enabled?
;Get current horiz position
;Branch if video echo
;If CH >= PWIDTH, then CH = COL
;Must be > col for valid tab
;Branch if ok
;8 or 16?
;If > forget it
;Find next comma cheaply
;Don't blame me it's Dick's trick
;Save the new position
;If col= width go back to start of line
;Go back to left edge
;Have we exceeded width?
;Are we tabbing?
;Space * 2
;CR * 2
;C = High bit

```

**:Communications port @ \$C200**

```

C25F:      C25E      61 testbhd equ *
C25E:68      C25F:20 70 CC      62      pla
C25F:20 70 CC      63      jsr update
C262:10 1B      C27F:      64      bpl serin
C264:20 A9 C7      65      jsr swcmd
C267:80 E3      C254      66      bcs noesc
C269:29 5F      67      and #55f
C26B:C9 51      68      cmp #0
C26D:F0 04      C273      69      beq exitx
C26F:C9 52      70      cmp #R'
C271:D0 09      C27C      71      bne term1
C273:A9 98      72      lda lda #598
C275:7A      73      exitx
C276:FA      74      plx
C277:60      75      rts
C278:18      76      goto mode
C279:20 A3 C7      77      goto term1
C27C:      C27C:20 4C CC      78      jsr showcur
C27F:48      79      pha
C280:20 D9 C7      80      serin
C283:80 09      C28E      81      snokbhd
C285:80 B8 06      82      bcs sidata
C288:29 10      83      lda flags,x
C28A:F0 D2      84      and #510
C28C:80 F2      C25E      85      beq testbhd
C28E:A8      C280      86      bra snokbhd
C28F:68      87      sida
C290:5A      88      pla
C291:20 B8 C3      89      pha
C294:68      90      jsr storh
C295:8C 38 06      91      pla
C298:F0 12      C2AC      92      ldy eschar,x
C29A:09 80      93      beq esnomod
C29C:C9 91      94      ora #80
C29E:F0 DC      95      cmp #xon
C2A0:C9 FF      96      beq term1
C2A2:F0 D8      97      cmp #5FF
C2A4:C9 92      98      beq term1
C2A6:F0 D0      99      cmp #592
C2A8:C9 94      100      beq goterm
C2AA:F0 C3      101      cmp #594
C2AC:3C B8 03      102      beq goterm
C2AF:50 A4      103      sinomod
C2B1:20 E0 FD      104      bvc exit1
C2B4:80 C6      105      jsr cout
C2B6:      C27C      106      jsr term1
C2B8:20 A0 CF      107      equ *
C2BA:80 29 C2      108      jsr movelrq
C2BC:20 7C C3      109      ldy defidx-$C1,x
C2BF:48      110      jsr getalt
C2C0:88      111      pha
C2C1:30 04      112      dey
C2C3:D0 03      113      bml
C2C5:D0 F5      114      cpy #3
C2C7:20 A0 CF      115      bne defloop
C2C9:68      116      jsr movelrq
C2CB:8C 2B C2      117      pla
C2CD:8C 2B C2      118      ldy devno,x
;Get current char
;Update cursor & check keyboard
;N=0 if no new key
;Test for command
;Branch if not
;upshift for following tests
;Quit?
;Reset?
;Go check serial
;return a CTRL-X
;Into remote mode
;Into terminal mode
;Get current char on screen
;Is it ready?
;Branch if we got data
;Is keyboard enabled?
;Branch if enabled
;Go test acia again
;Save new input in y for now
;Save new char on stack
;Fix the screen
;Get the new data
;If 0, don't modify char
;Apple loves the high bit
;Ignore ^Q
;Ignore rfs
;N for remote?
;T for terminal mode?
;In terminal mode?
;Return to user if not A = char
;Onto the screen with it
;Set up the defaults
;make sure irq vectors ok
;Index into alt screen. Table in command
;Get default from alt screen
;Done if minus
;Or if 2
;Jam irq vector into LC
;Command, control & flags on stack

```

06 COMM	Communications port routine	20-OCT-86 06:41 PAGE 15	07 C3SPACE	Communications port routine	20-OCT-86 06:41 PAGE 16
C2CE:99 FB BF	119 sta scntl,y		C300:	2 *****	
C2D1:68	120 pla	;Set command reg	C300:	3 *	
C2D2:99 FA BF	121 sta scomd,y		C300:	4 * THIS IS THE \$C3XX ROM SPACE:	
C2D5:68	122 pla		C300:	5 *	
C2D6:9D B8 06	123 sta flags,x	;And the flags	C300:	6 *****	
C2D9:29 01	124 and #1	;A = \$01 (*A) if comm mode	C300:48	7 C3ENTRY PHA	;save regs
C2D8:00 02	125 bne defcom		C301:DA	8 PHX	
C2D0:A9 09	126 lda #9	;1 for serial port	C302:5A	9 PHX	
C2D8:9D 38 06	127 defcom		C303:80 12 C317	10 BRA BASICINIT	;and init video firmware
C2E2:68	128 sta eschar,x		C305:38	11 C3KEYIN SEC \$90	;Pascal 1.1 ID byte
C2E3:9D B8 04	129 sta pwidth,x	;Get printer width	C306:90	12 C3COUTI DFB	;BCC OP CODE (NEVER TAKEN)
C2E6:9E B8 03	130 stz sermode,x		C307:18	13 C3COUTI CLC	;Pascal 1.1 ID byte
C2E9:60	131 rts		C308:80 1A C324	14 BRA BASICENT	;=>go print/read char
C2EA:03 07	132 defidx		C30A:EA	15 NOP	
C2EC:	133 sldfidx		C30B:	16 *	
C2Z8	134 devno		C30B:	17 * PASCAL 1.1 FIRMWARE PROTOCOL TABLE:	
C2EC:AO B0	135		C30B:	18 *	
C2E9:	0012	ds \$C300+,\$90	C30B:01	19 DFB \$01	;GENERIC SIGNATURE BYTE
C300:	44	include c3space	C30C:88	20 DFB \$88	;DEVICE SIGNATURE BYTE
			C30D:	21 *	
			C30D:2C	22 DFB >JPINIT	;PASCAL INIT
			C30E:2F	23 DFB >JPREAD	;PASCAL READ
			C30F:32	24 DFB >JPWRITE	;PASCAL WRITE
			C310:35	25 DFB >JPSTAT	;PASCAL STATUS
			C311:	26 *****	
			C311:	27 *	
			C311:	28 * 128K SUPPORT ROUTINE ENTRIES:	
			C311:	29 *	
			C311:4C AF C7	30 JMP SNAUX	;MEMORY MOVE ACROSS BANKS
			C314:4C B5 C7	31 JMP SWXFER	;TRANSFER ACROSS BANKS
			C317:	32 *****	
			C317:	33 *	
			C317:	34 *****	
			C317:	35 * BASIC I/O ENTRY POINT:	
			C317:	36 *****	
			C317:	37 *	
			C317:20 20 CE	38 BASICINIT JSR HOOKUP	;COPYROM if needed, sethooks
			C31A:20 BE CD	39 JSR SET80	;setup 80 columns
			C31D:20 58 FC	40 JSR HOME	;clear screen
			C320:7A	41 PLY	
			C321:FA	42 PLA	;restore x
			C322:68	43 PLA	;restore char
			C323:18	44 CLC	;output a character
			C324:	45 *	
			C324:90 03 C329	46 BASICENT BCS BINPUT	;=>carry me to input
			C326:4C F6 FD	47 BPRINT JMP COUTE	;print a character
			C329:4C 1B FD	48 BINPUT JMP KEYIN	;get a keystroke
			C32C:	49 *	
			C32C:4C 41 CF	50 JPINIT JMP PINIT	;pascal init
			C32F:4C 35 CF	51 JPREAD JMP PASREAD	;pascal read
			C332:4C C2 CE	52 JPWRITE JMP PWRITE	;pascal write
			C335:4C B1 CE	53 JPSTAT JMP PSTATUS	;pascal status call
			C338:	54 *	
			C338:	55 * COPYROM is called when the video firmware is	
			C338:	56 * initialized. If the language card is switched	
			C338:	57 * in for reading, it copies the F8 ROM to the	
			C338:	58 * language card and restores the state of the	
			C338:	59 * language card.	

```

C338: 60 *
C339:A9 06 61 COPROM LDA #GOODP8 ;get the ID byte
C33A: 62 *
C33A: 63 * Compare ID bytes to whatever is readable. If it
C33A: 64 * matches, all is ok. If not, need to copy.
C33A: 65 *
C33A: 66 * CMP FVERSION ;does it match?
C33A:CD B3 FB 67 BRQ ROMOK
C33D:F0 3C C37B 68 JSR SETROM ;read ROM, write RAM, save state
C33F:20 60 C3 69 LDA #F8 ;from F800-FFFF
C342:A9 F8 70 STA CSWE
C344:85 37 71 STB CSWL
C346:64 36 72 COPROM2 LDA (CSWL) ;get a byte
C348:B2 36 73 STA (CSWL) ;and save a byte
C34A:92 36 74 INC CSWL
C34C:E6 36 75 BNE COPROM2
C34E:D0 F8 C348 76 INC CSWE
C350:26 37 77 BNE COPROM2 ;fall into RESETLIC
C352:D0 F4 C348 78 *
C354: 79 * RESETLIC resets the language card to the state
C354: 80 * determined by SETROM. It always leaves the card
C354: 81 * write enabled.
C354: 82 *
C354: 83 RESETLIC PEX ;save X
C355:AE 78 04 84 LDX ROMSTATE ;get the state
C358:3C 81 C0 85 BIT ROMIN,X ;set bank 1 ROM/RAM read
C358:3C 81 C0 86 BIT ROMIN,X ;set write enable
C35E:7A 87 PLX ;restore X
C35F:60 88 RTS
C360: 89 *
C360: 90 * SETROM switches in the ROM for reading, the RAM
C360: 91 * for writing, and it saves the state of the
C360: 92 * language card. It does not save the write
C360: 93 * protect status of the card.
C360: 94 *
C360: 95 SETROM PEX ;save X
C361:A2 00 96 LDX #0 ;assume write enable, bank2, ROMRD
C363:2C 11 C0 97 BIT RDCHNK2 ;is bank 2 switched in?
C366:30 02 C36A 98 BMT NOT1 ;=>yes
C368:A2 08 99 LDX #8 ;indicate bank 1
C36A:2C 12 C0 100 NOT1 ;is LC RAM readable?
C36D:10 02 C371 101 BPL NOREAD ;=>no
C36F:E8 102 INX ;indicate RAM read
C370:E8 103 INX
C371:2C 81 C0 104 NOREAD BIT $C081 ;ROM read
C374:2C 81 C0 105 BIT $C081 ;RAM write
C377:8E 78 04 106 STX ROMSTATE ;save state
C37A:FA 107 PLX ;restore X
C37B:60 108 ROMOK RTS
C37C: 109 *
C37C: 110 * GETALT reads a byte from aux memory screenholes.
C37C: 111 * Y is the index to the byte [0-7] indexed off of
C37C: 112 * address $478.
C37C: 113 *
C37C:AD 13 C0 114 GETALT LDA ROMRMD ;save state of aux memory
C37F:0A 115 ASL A
C380:AD 18 C0 116 LDA ROMCOL ;and of the 80STORE switch
C383:08 117 PEP

```

```

C384:80 03 C0 118 STA C380COL ;no 80STORE to get page 1
C387:80 03 C0 119 STA ROCARDRAM ;pop in the other half of RAM
C38A:89 78 04 120 LDA $478,Y ;read the desired byte
C38D:28 121 PIP ;and restore memory
C38E:03 C393 122 BCS GETALT1
C390:80 02 C0 123 STA ROMAINRAM
C393:10 03 C398 124 GETALT1 BPL GETALT2
C395:80 01 C0 125 STA SET80COL
C398:60 126 GETALT2 RTS
C399: 127 *
C399:09 80 128 UPSHIFT0 ORA #80 ;set high bit for exers
C39B:C9 FB 129 UPSHIFT CMP #F8
C39D:80 06 C3A5 130 BCS X.UPSHIFT
C39F:C9 E1 131 CMP #E1
C3A1:90 02 C3A5 132 BCC X.UPSHIFT
C3A3:29 DF 133 AND #5DF
C3A5:60 134 X.UPSHIFT RTS
C3A6: 135 *
C3A6: 136 * GETCOUT performs COUT for GETIN. It disables the
C3A6: 137 * echoing of control characters by clearing the
C3A6: 138 * M.CTL mode bit, prints the char, then restores
C3A6: 139 * M.CTL. NOESC is used by the RKEY routine to
C3A6: 140 * disable escape sequences..
C3A6: 141 *
C3A6: 142 GETCOUT PEA ;save char to print
C3A7:A9 08 143 LDA #M.CTL ;disable control chars
C3A9:1C FB 04 144 STB WMODE ;by clearing M.CTL
C3AC:68 145 PLA ;restore character
C3AD:20 ED FD 146 JSR COUT ;and print it
C3B0:4C 44 FD 147 JMP NOESCAPE ;enable control chars
C3B3: 148 *
C3B3: 149 * STORCH determines loads the current cursor position,
C3B3: 150 * inverts the character, and displays it
C3B3: 151 * STORCH inverts the character and displays it at the
C3B3: 152 * position stored in Y
C3B3: 153 * STORY determines the current cursor position, and
C3B3: 154 * displays the character without inverting it
C3B3: 155 * STORE displays the char at the position in Y
C3B3: 156 *
C3B3: 157 * If mouse characters are enabled (WMODE bit 0 = 0)
C3B3: 158 * then mouse characters ($40-$5F) are displayed when
C3B3: 159 * the alternate character set is switched in. Normally
C3B3: 160 * values $40-$5F are shifted to $0-$1F before display.
C3B3: 161 *
C3B3: 162 * Calls to GETCUR trash Y
C3B3: 163 *
C3B3: 164 STORY JSR GETCUR ;get newest cursor into Y
C3B6:80 09 C3C1 165 BRA STORE
C3B8: 166 *
C3B8:20 90 CC 167 STORCH JSR GETCUR ;first, get cursor position
C3B8:24 32 168 BIT INWIG ;normal or inverse?
C3B8:30 02 C3C1 169 BMT STORE ;=>normal, store it
C3BF:29 7F 170 AND #57F ;inverse it
C3C1:5A 171 STORE PHX ;save real Y
C3C2:09 00 172 ORA #0 ;does char have high bit set?
C3C4:30 15 C3DB 173 BMT STORE1 ;=>yes, don't do mouse check
C3C6:48 174 PEA ;save char
C3C7:AD FB 04 175 LDA WMODE ;is mouse bit set?

```

07 C3SPACE		20-OCT-86 06:41 PAGE 19		20-OCT-86 06:41 PAGE 20	
Communications port routine		08 EQUATES2		slinky equates	
C3CA:6A	176	ROR A	C400:	2 *****	
C3CB:68	177	PIA	C400:	3 * slinky equates	
C3CC:90 0D C3DB	178	BCC STORE1	C400:	4 *****	
C3CE:9C 1E C0	179	BIT ALTCHARSET	C400:	5 revnum equ \$101	;revision 1.0.1
C3D1:10 08 C3DB	180	BPL STORE1	C400:	6 pcrevnum equ \$11	;smartport rev 1.1
C3D3:49 40 181	181	EOR \$F40			
C3D5:89 60 182	182	BIT \$F60			
C3D7:F0 02 C3DB	183	BEQ STORE1	C400:	8 * prodos equates	
C3D9:49 40 184	184	EOR \$F40			
C3DB:2C 1F C0	185	STORE1	C400:	10 cmand equ \$42	;command to be executed
C3DE:10 19 C3F9	186	BIT RDRVID	C400:	11 unit equ \$43	;B<6-4> = slot
C3E0:48 187	187	BPL STORE5	C400:	12 buffer equ \$44	;pointer to 512 byte data buffer
C3E1:80 01 C0	188	STA SET80COL	C400:	13 block equ \$46	;block number
C3E4:98 189	189	TVA			
C3E5:45 20 190	190	EOR WNDLFT	C400:	15 * protocol converter equates	
C3E7:4A 191	191	LSR A	C400:	17 pparam equ \$43	;parameter count
C3E8:80 04 C3EE	192	BCS STORE2	C400:	18 punit equ \$44	;unit number
C3EA:AD 55 C0	193	LDA TXTPAGE2	C400:	19 pbuff equ \$45	;two byte buffer pointer
C3ED:C8 194	194	INV	C400:	20 pstat equ \$47	;status / control code
C3EE:98 195	195	STORE2	C400:	21 pblock equ \$47	;block number
C3EF:4A 196	196	LSR A	C400:	22 pcount equ \$47	;byte count
C3F0:A8 197	197	TAY	C400:	23 paddr equ \$49	;address for read
C3F1:68 198	198	PIA	C400:	24 temptr equ \$4A	;pointer to params must be last 2 zp byte
C3F2:91 28	199	STORE3	C400:	25 zused equ	temptr-cmand+2 ;zero page bytes used
C3F4:2C 54 C0	200	BIT TXTPAGE1			
C3F7:7A 201	201	STORE4			
C3F8:60 202	202	RTS			
C3F9: 203 *	203		C400:	27 * prodos commands	
C3F9:91 28	204	STA (BASL),Y	C400:	29 prostat equ 0	;status command
C3FB:7A 205	205	PLY	C400:	30 proread equ 1	;read command
C3FC:60 206	206	RTS	C400:	31 prowrite equ 2	;write command
C400: 0003 207	207	DS	C400:	32 proform equ 3	;format command
	45	Include equates2			
			C400:	34 * DOS equates	
			C400:	36 iobpl equ \$48	;pointer to IOB
			C400:	37 iobph equ \$49	
			C400:	38 lsslot equ 1	;slot * 16
			C400:	39 lbrvrv equ 2	;drive 1 or 2
			C400:	40 lbrtrk equ 4	;track number
			C400:	41 lbrsect equ 5	;sector number
			C400:	42 lbrbuff equ 8	;buffer pointer
			C400:	43 lbrcmd equ 12	;command
			C400:	44 lbrstat equ 13	;status
			C400:	45 doserr equ \$80	;DOS I/O error
			C400:	9D1E 46 dosinit equ \$901E	;DOS init vector use addr-1
			C400:	A6C3 47 dossyn equ \$A6C4-1	;DOS syntax error
			C400:	BD00 48 rwt5 equ \$BD00	;RWTS entry point
			C400:	50 * error codes	
			C400:	52 badcmd equ \$01	;bad command
			C400:	53 badpct equ \$04	;bad parameter count
			C400:	54 badunit equ \$11	;bad unit number
			C400:	55 badctl equ \$21	;bad control / status code
			C400:	56 loerr equ \$27	;other I/O error
			C400:	57 nderr equ \$28	;no device error
			C400:	58 badblk equ \$2D	;bad block or address

08 EQUATES2

slinky equates

09 SLINKY

slinky entry points

```

C400:
C400: 0002 62 bootjmp equ 2 ;make jmp to entry
C400: 0000 63 bootblk equ 0 ;reads block 0
C400: 0800 64 bootbuf equ $800 ;into $800
C400: FABA 65 autoscan equ $FABA ;re-entry point to auto boot
C400: FFS9 66 monitor equ $FFS9 ;go to monitor if boot fails

C400:
C400: 68 * scratch area equates
C400: 0478 70 sizetemp equ $478 ;holds # blocks
C400: 0478 71 error equ $478 ;error flag
C400: 0578 72 xval equ $578 ;value to be returned in X
C400: 0578 73 yval equ $578 ;value to be returned in Y
C400: 0678 74 sl.lststate equ $678 ;language card state
C400: 0778 75 sl.devno equ $778 ;slot * 16 ($n0) + $88
C400: 0778 76 sl.slot equ $778 ;$C0 + slot ($Cn)

C400:
C400: 78 * slot ram equates
C400: 0388 80 sl.scrn1 equ $478-$C0
C400: 0438 81 sl.scrn2 equ $478-$C0
C400: 0488 82 sl.scrn3 equ $578-$C0
C400: 0538 83 sl.scrn4 equ $578-$C0
C400: 0588 84 sl.scrn5 equ $678-$C0
C400: 0638 85 sl.scrn6 equ $678-$C0
C400: 0688 86 sl.scrn7 equ $778-$C0
C400: 0738 87 sl.scrn8 equ $778-$C0
C400: 0388 88 numbanks equ sl.scrn1
C400: 0688 89 powerup equ sl.scrn7
C400: 0400: 90 ;power2 equ sl.scrn8

C400:
C400: 92 * hardware equates, must be in $BFF0 to avoid double access
C400: BFF8 94 addr1 equ $BFF8 ;address pointer
C400: BFF9 95 addr2 equ $BFF9 ;auto incs after every data access
C400: BFFA 96 addr3 equ $BFFA ;data pointed to
C400: BFFB 97 data equ $BFFB

C400:
C400: 99 * other interface equates
C400: BFF0 101 proflag equ $BFF0 ;0 = Pascal, $4C = ProDOS, other = DOS
C400: 00AA 102 nameflg equ $AA ;value unused in any catalog
C400: 00FC 103 sizeflg equ $FC ;block size flag
C400: 00FD 104 zers equ $FD ;catalog skip flag
C400: 00FE 105 skipfe equ $FE ;skip FE bytes in catalog
C400: 0004 106 slot equ $04 ;slot #
C400: DC00 107 diagcode equ $DC00 ;location of diagnostic code
C400: 2000 108 diagdest equ $2000 ;location of diagnostics in ram
C400: 1FFF 109 diagstart equ $1FFF ;start location of diagnostics
C400: 46 include slinky ;ram card at $C400

```

```

C400: 2 *****
C400: 3 * slinky boot code *****
C400: 4 *****
C400: 6 cmp #520 ;Boot entry point
C400: 7 cmp #500 ;Signature byte stuff
C400: 8 cmp #903
C400: 9 cmp #0 ;Always taken
C400: 10 bcs boot4 ;Diagnostics entry point
C400: 11 ldy #5
C400: 12 bne dos24
C400: 13 * Here is the boot code
C400: 14 * Reads in block 0 into $800 and executes at $801
C400: 15 boot4 equ *
C400: 16 sei ;No interrupts if booting
C400: 17 lda kswh ;Save IN#
C400: 18 *****
C400: 19 *****
C400: 20 * the following two bytes must be $90 and $4B in locations $C411 and
C400: 21 * and $C412 respectively. the bcc ($90) is never taken by the
C400: 22 * slinky code and the $4B is used to duplicate the mouse entry
C400: 23 * point as found in slot 7. this 'fix' enables some programs
C400: 24 * to still work correctly. (Cim, you owe me a beer for this one!)
C400: 25 *****
C400: 27 ddb $90,$4B ;bcc is never taken
C400: 29 pha
C400: 30 jsr setbld ;Reset the hooks
C400: 31 jsr setvid
C400: 32 pla
C400: 33 jsr slboot ;Go get boot block
C400: 34 ldx bootbuf+1
C400: 35 beq btok4 ;boot not okay
C400: 36 ldx #4*$10 ;X=n0
C400: 37 jmp bootbuf+1
C400: 38 *****
C400: 39 * discontinue boot sequence if not power on reset or forced cold start
C400: 41 btok4 lda loc0
C400: 42 bne btok4.1
C400: 43 lda loc1
C400: 44 cmp #4*$C0
C400: 45 bne btok4.1
C400: 46 lda #5C6
C400: 47 sta loc1 ;should be $C6 now
C400: 48 jmp (loc0) ;try slot 6 instead
C400: 49 btok4.1 lda #23 ;go to bottom of screen
C400: 50 sta cv
C400: 51 *****
C400: 53 btok4.2 lda utsmgs,x
C400: 54 beq btok4.3 ;'unable to start from memory card.'
C400: 55 jsr cout ;skip if done
C400: 56 lnx
C400: 57 bne btok4.2
C400: 58 *****
C400: 59 btok4.3 jmp basic ;drop into basic

```

C44B:4C 1C C7 61 jmp xsetmou ;should be at SC44B

```

slinky entry points
C44E: *****
C44E: * Entry point for prodos driver
C44E: *****

C44E:4C 54 C4 67 entry4 jmp ent4 ;Jump to ProDOS
C451:4C 94 C4 68 jmp pconv4 ;Jump to PC
C454:A9 28 69 ent4 lda #derr ;Assume wrong drive
C456:A6 43 70 ldx unit
C458:30 2A C484 71 bml rats4 ;Error!!!
C45A:A9 01 72 lda #badcmd ;Assume bad command
C45C:A4 42 73 ldy cmdand ;Get command

C45E: *****
C45E: * the 'copy #' below use to be a 'cmp #' because of this
C45E: * change, revnum will be 1.0.1

C45E:C0 04 78 copy #4
C460:B0 22 C484 79 bpe rats4 ;Branch if bad command
C462:A2 0A 80 dos24 ldx #rused ;Save zp DOS jumps in here
C464:B5 41 81 zsave4 lda cmdand-1,x
C466:48 82 pha
C467:CA 83 dex
C468:D0 FA C464 84 bne zsave4
C46A:98 85 tja
C46B:18 86 clc
C46C:69 14 87 adc #20 ;Add 20 to command for table look up

C46E: *****
C46E: * This code is common to PC and ProDOS and DOS entry points
C46E: *****

C46E:20 52 C7 93 doit4 jar s1req ;Go do command
C471:A2 00 94 done4 ldx #0 ;Restore zp
C473:68 95 rslloop4 pla cmdand,x
C474:95 42 96 sta
C476:E8 97 inx
C477:E0 0A 98 cpx #rused
C479:90 F8 C473 99 bit rslloop4
C47B:AE 78 05 100 ldx xval
C47E:AC F8 05 101 ldy yval
C481:AD F8 04 102 lda error
C484:C9 01 103 rats4 cmp #1 ;Was there an error?
C486:09 00 104 ora #0 ;C = 1 if not 0
C488:60 105 rts ;Set n,z flags

C489:A9 01 107 pcmd4 lda #badcmd ;Bad command
C48B:D0 02 C48F 108 bne perr4 ;Bad protocol unit
C48D:A9 11 109 pbad4 lda #badunit
C48F:8D F8 04 110 perr4 sta error ;Save the error code
C492:D0 D0 C471 111 bne done4

```

09 ST.INKY

```

C401:
156 *****
157 * DOS entry point stuff
158 *****
159 *****
160 dosent4      sty      iobpl      ;Store pointer to IOB
161              sta      iobph      ;Get slot
162              ldy      fibslot     ;Is it us?
163              lda      [iobpl],y   ;Back to ruts
164              cmp      #4*$10      ;Command for DOS
165              beq      dosent4     ;Go do it and come back
166              jmp      ruts+4      ;A = Error code
167              ldy      #4          ;DOS io error code
168              jsr      dos24        ;Put error code in status
169              beq      doserr      ;
170              lda      doserr      ;
171              ldy      fibstat     ;
172              sta      [iobpl],y   ;
173              rts
174
175              ds      $C4FA-*,0
176
177              db      pcrevnum
178              db      $01
179              dw      0
180              db      $AF
181              db      $entry4
182              include misc
183              ds      $C48E-*,0
184
185              or8E    1
186              ds      $C500-*,0

```

```

C94:      113 *****
C94:      114 * Protocol converter entry point
C94:      115 *****
C94:
C94:      117 pconv4   pla      ;Pull the return address
C94:      118          tay
C94:      119          cmp      $STD
C94:      120          pla      ;C = 1 if carry in +3
C94:      121          tax      ;High byte of return address
C94:      122          adc      #0
C94:      123          pha
C94:      124          tya
C94:      125          adc      #3
C94:      126          lda      temptr+1
C94:      127          pha
C94:      128          pha
C94:      129          stx      temptr+1
C94:      130          ldx      #used-1
C94:      131          lda      pcsvzp4
C94:      132          pha
C94:      133          dex
C94:      134          bne      pcsvzp4
C94:      135          sty      temptr
C94:      136          ldy      #3
C94:      137          pcvt4
C94:      138          lda      (temptr),y
C94:      139          dey
C94:      140          bne      pcvt4
C94:      141          tax
C94:      142          ;
C94:      143          ldy      #8
C94:      144          pcparms4
C94:      145          sta      (command),y
C94:      146          dey
C94:      147          bpl      pcparms4
C94:      148          lsr      punit
C94:      149          bne      pcbad4
C94:      150          txa
C94:      151          rol      A
C94:      152          cmp      #20
C94:      153          bcc      pccmd4
C94:      154          bcs      doit4
C94:      155          ;
C94:      156          ;
C94:      157          ;
C94:      158          ;
C94:      159          ;
C94:      160          ;
C94:      161          ;
C94:      162          ;
C94:      163          ;
C94:      164          ;
C94:      165          ;
C94:      166          ;
C94:      167          ;
C94:      168          ;
C94:      169          ;
C94:      170          ;
C94:      171          ;
C94:      172          ;
C94:      173          ;
C94:      174          ;
C94:      175          ;
C94:      176          ;
C94:      177          ;
C94:      178          ;
C94:      179          ;
C94:      180          ;
C94:      181          ;
C94:      182          ;
C94:      183          ;
C94:      184          ;
C94:      185          ;
C94:      186          ;
C94:      187          ;
C94:      188          ;
C94:      189          ;
C94:      190          ;
C94:      191          ;
C94:      192          ;
C94:      193          ;
C94:      194          ;
C94:      195          ;
C94:      196          ;
C94:      197          ;
C94:      198          ;
C94:      199          ;
C94:      200          ;
C94:      201          ;
C94:      202          ;
C94:      203          ;
C94:      204          ;
C94:      205          ;
C94:      206          ;
C94:      207          ;
C94:      208          ;
C94:      209          ;
C94:      210          ;
C94:      211          ;
C94:      212          ;
C94:      213          ;
C94:      214          ;
C94:      215          ;
C94:      216          ;
C94:      217          ;
C94:      218          ;
C94:      219          ;
C94:      220          ;
C94:      221          ;
C94:      222          ;
C94:      223          ;
C94:      224          ;
C94:      225          ;
C94:      226          ;
C94:      227          ;
C94:      228          ;
C94:      229          ;
C94:      230          ;
C94:      231          ;
C94:      232          ;
C94:      233          ;
C94:      234          ;
C94:      235          ;
C94:      236          ;
C94:      237          ;
C94:      238          ;
C94:      239          ;
C94:      240          ;
C94:      241          ;
C94:      242          ;
C94:      243          ;
C94:      244          ;
C94:      245          ;
C94:      246          ;
C94:      247          ;
C94:      248          ;
C94:      249          ;
C94:      250          ;
C94:      251          ;
C94:      252          ;
C94:      253          ;
C94:      254          ;
C94:      255          ;
C94:      256          ;
C94:      257          ;
C94:      258          ;
C94:      259          ;
C94:      260          ;
C94:      261          ;
C94:      262          ;
C94:      263          ;
C94:      264          ;
C94:      265          ;
C94:      266          ;
C94:      267          ;
C94:      268          ;
C94:      269          ;
C94:      270          ;
C94:      271          ;
C94:      272          ;
C94:      273          ;
C94:      274          ;
C94:      275          ;
C94:      276          ;
C94:      277          ;
C94:      278          ;
C94:      279          ;
C94:      280          ;
C94:      281          ;
C94:      282          ;
C94:      283          ;
C94:      284          ;
C94:      285          ;
C94:      286          ;
C94:      287          ;
C94:      288          ;
C94:      289          ;
C94:      290          ;
C94:      291          ;
C94:      292          ;
C94:      293          ;
C94:      294          ;
C94:      295          ;
C94:      296          ;
C94:      297          ;
C94:      298          ;
C94:      299          ;
C94:      300          ;
C94:      301          ;
C94:      302          ;
C94:      303          ;
C94:      304          ;
C94:      305          ;
C94:      306          ;
C94:      307          ;
C94:      308          ;
C94:      309          ;
C94:      310          ;
C94:      311          ;
C94:      312          ;
C94:      313          ;
C94:      314          ;
C94:      315          ;
C94:      316          ;
C94:      317          ;
C94:      318          ;
C94:      319          ;
C94:      320          ;
C94:      321          ;
C94:      322          ;
C94:      323          ;
C94:      324          ;
C94:      325          ;
C94:      326          ;
C94:      327          ;
C94:      328          ;
C94:      329          ;
C94:      330          ;
C94:      331          ;
C94:      332          ;
C94:      333          ;
C94:      334          ;
C94:      335          ;
C94:      336          ;
C94:      337          ;
C94:      338          ;
C94:      339          ;
C94:      340          ;
C94:      341          ;
C94:      342          ;
C94:      343          ;
C94:      344          ;
C94:      345          ;
C94:      346          ;
C94:      347          ;
C94:      348          ;
C94:      349          ;
C94:      350          ;
C94:      351          ;
C94:      352          ;
C94:      353          ;
C94:      354          ;
C94:      355          ;
C94:      356          ;
C94:      357          ;
C94:      358          ;
C94:      359          ;
C94:      360          ;
C94:      361          ;
C94:      362          ;
C94:      363          ;
C94:      364          ;
C94:      365          ;
C94:      366          ;
C94:      367          ;
C94:      368          ;
C94:      369          ;
C94:      370          ;
C94:      371          ;
C94:      372          ;
C94:      373          ;
C94:      374          ;
C94:      375          ;
C94:      376          ;
C94:      377          ;
C94:      378          ;
C94:      379          ;
C94:      380          ;
C94:      381          ;
C94:      382          ;
C94:      383          ;
C94:      384          ;
C94:      385          ;
C94:      386          ;
C94:      387          ;
C94:      388          ;
C94:      389          ;
C94:      390          ;
C94:      391          ;
C94:      392          ;
C94:      393          ;
C94:      394          ;
C94:      395          ;
C94:      396          ;
C94:      397          ;
C94:      398          ;
C94:      399          ;
C94:      400          ;
C94:      401          ;
C94:      402          ;
C94:      403          ;
C94:      404          ;
C94:      405          ;
C94:      406          ;
C94:      407          ;
C94:      408          ;
C94:      409          ;
C94:      410          ;
C94:      411          ;
C94:      412          ;
C94:      413          ;
C94:      414          ;
C94:      415          ;
C94:      416          ;
C94:      417          ;
C94:      418          ;
C94:      419          ;
C94:      420          ;
C94:      421          ;
C94:      422          ;
C94:      423          ;
C94:      424          ;
C94:      425          ;
C94:      426          ;
C94:      427          ;
C94:      428          ;
C94:      429          ;
C94:      430          ;
C94:      431          ;
C94:      432          ;
C94:      433          ;
C94:      434          ;
C94:      435          ;
C94:      436          ;
C94:      437          ;
C94:      438          ;
C94:      439          ;
C94:      440          ;
C94:      441          ;
C94:      442          ;
C94:      44
```

```

20-OCT-86 06:41 PAGE 27
slinky entry points
10 MISC
C58E:
C58E:
C58E:
C58E:A2 03 7 MXTBL LDX #503
C590:A0 00 8 LDX #0
C592:86 3C 9 TELL00P STX BOOTMP
C594:8A 10 10 TXA
C595:0A 11 ASL A
C596:24 3C 12 BIT BOOTMP
C598:F0 10 13 BEQ NOPATRN
C59A:05 3C 14 ORA BOOTMP
C59C:49 FF 15 EOR #5F
C59E:29 7E 16 AND #7E
C5A0:80 08 C5AA 17 TELL00P2 ECS NOPATRN
C5A2:4A 18 LSR A
C5A3:D0 FB C5A0 19 ENE TELL00P2
C5A5:98 20 TTA
C5A6:9D 56 03 21 STA DNIBL,X
C5A9:C8 22 INY
C5AB:E8 23 NOPATRN INX
C5AB:10 E5 C592 24 BPL TELL00P
C5AD:A9 08 25 LDA #508
C5AF:85 27 26 STA $27
C5B1:A0 7F 27 LDY #7F
C5B3:60 28 RTS
C5B4:
C5B4:
C5B4:
C5B4:B9 00 02 34 getup lda in,y :get character
C5B7:C8 35 iny
C5B8:4C 99 C3 36 jmp upshift0
C5BB:
C5BB:
C5BB:
C5BB:
C5BB:C1 F0 F0 EC 42 MSB ON
C5C4: 43 apple2c asc 'Apple
C5C4: 44 MSB OFF //c'
C5C4:
C5C4:
C5C4:
C5C4:20 D0 F8 46 *****
C5C7:20 53 F9 47 * showinst - disassemble an instruction and adjust the PC
C5CA:85 3A 48 *****
C5CC:84 3B 49 *****
C5CE:60 50 *****
C5C7: 51 *****
C5C7: 52 *****
C5C7: 53 *****
C5C7: 54 *****
C5C7: 55 *****
C5C7: 56 *****
C5C7: 57 *****
C5C7: 58 *****
C5C7:5A 60 *****

```

```

20-OCT-86 06:41 PAGE 28
slinky entry points
10 MISC
C5D0:B0 1C C5EE 61 bcs gobasicin
C5D2:A0 C7 62 ldy #basic
C5D4:C4 39 63 cpy kwh
C5D6:D0 04 C5DC 64 bne xmbout
C5D8:A4 38 65 ldy kwl
C5DA:F0 12 C5EE 66 beq gobasicin
C5DD:48 67 xmbout phx
C5DE:29 7F 68 pha
C5E0:C9 02 69 and #7F
C5E2:B0 06 C5EA 70 cmp #2
C5E4:20 1C C7 71 bge mbbad
C5E7:20 3A C7 72 jsr xsetmou
C5EA:68 73 jsr xmbome
C5EB:FA 74 mbbad pla
C5EC:7A 75 plx
C5ED:60 76 ply
C5EE:4C 9D C7 77 rts
C5F1: 80 ds $C5F5-+0
C5F3: 48 include boot
C5F5: 000B 1 ds $C600-+0

```

```

;input?
;input from $C400?
;save X too
;we don't care about high bit
;only 0,1 valid
;go to input routine
;more disk stuff
;Disk II boot $9C600
;Disk II in slot 6

```

11 BOOT

```

C600: 4 *****
C600: 5 *
C600: 6 * Disk II boot stuff
C600: 7 * jumps to slot 5 if boot fails
C600: 8 *
C600: 9 *****
C600: 10 LDX #520
C600: 11 LDY #500
C600: 12 STZ $03
C600: 13 STZ $3C
C600: 14 LDA #560
C600: 15 TAX
C600: 16 DRV2ENT STX SLOTTZ
C600: 17 STA BOOTDEV
C600: 18 PHY
C600: 19 LDA SC08E,X
C600: 20 LDA SC08C,X
C600: 21 PLY
C600: 22 LDA SC0EA,Y
C600: 23 LDA SC089,X
C600: 24 LDY #550
C600: 25 SEEKZERO LDA SC080,X
C600: 26 TTY AND #503
C600: 27 AND ASL A
C600: 28 ORA SLOTTZ
C600: 29 TAX
C600: 30 LDA SC081,X
C600: 31 LDA #556
C600: 32 LDX LDA #556
C600: 33 JSR WAIT
C600: 34 DEV
C600: 35 RPL SEEKZERO
C600: 36 STA $26
C600: 37 STA $3D
C600: 38 STA $41
C600: 39 JSR maktbl
C600: 40 EXTENT1 STZ $03
C600: 41 RMDR CLC
C600: 42 PHP
C600: 43 RETRY1 PLY
C600: 44 RMDR LDX
C600: 45 DEC
C600: 46 BNE RMDR0
C600: 47 FUGIT LDA SC088,X
C600: 48 lda loc1
C600: 49 cmp #56
C600: 50 bne bootfail
C600: 51 jmp $c500
C600: 53 ds $C656-*,0
C600: 54 RMDR0 PHP
C600: 55 RETRY DEV
C600: 56 BNE RMDR1
C600: 57 REQ RETRY1
C600: 58 EXTENT BRA EXTENT1
C600: 59 *****
C600: 60 * The following code is sacred in it's *
C600: 61 * present form. To change it would *

```

Disk II boot code

11 BOOT

```

C652: 62 * cause volcanos to erupt, the ground *
C652: 63 * to shake, and ProDOS not to boot! *
C652: 64 *****
C652: 65 RMDR1 LDA SC08C,X
C652: 66 BPL RMDR1
C652: 67 ISMRK1 EOR #5D5
C652: 68 BNE RETRY
C652: 69 RMDR2 LDA SC08C,X
C652: 70 BPL RMDR2
C652: 71 CMP #5AA
C652: 72 BNE ISMRK1
C652: 73 NOP
C652: 74 RMDR3 LDA SC08C,X
C652: 75 BPL RMDR3
C652: 76 CMP #596
C652: 77 BEQ RMDSECT
C652: 78 PLY
C652: 79 BCC RMDR
C652: 80 EOR #5AD
C652: 81 BEQ RDATA
C652: 82 BNE RMDR
C652: 83 RMDSECT LDY #503
C652: 84 RMDSECT1 STA $40
C652: 85 RMDSECT2 LDA SC08C,X
C652: 86 BPL RMDSECT2
C652: 87 ROL A
C652: 88 STA BOOTTMP
C652: 89 RMDSECT3 LDA SC08C,X
C652: 90 BPL RMDSECT3
C652: 91 AND BOOTTMP
C652: 92 DEY
C652: 93 BNE RMDSECT1
C652: 94 PLY
C652: 95 CMP $3D
C652: 96 BNE RMDR
C652: 97 LDA $40
C652: 98 CMP $41
C652: 99 RMDR1 BNE RMDR
C652: 100 ECS RMDR
C652: 101 RDATA LDY #556
C652: 102 RDATA STY BOOTTMP
C652: 103 RDATA LDY SC08C,X
C652: 104 BPL RDATA1
C652: 105 EOR DN1EL-$80,Y
C652: 106 LDY BOOTTMP
C652: 107 DEY
C652: 108 STA NEUFI,Y
C652: 109 BNE RDATA
C652: 110 RDATA2 STY BOOTTMP
C652: 111 RDATA3 LDY SC08C,X
C652: 112 BPL RDATA3
C652: 113 EOR DN1EL-$80,Y
C652: 114 LDY BOOTTMP
C652: 115 STA ($26),Y
C652: 116 INY
C652: 117 RMDT2 BNE RDATA2
C652: 118 RDATA4 LDY SC08C,X
C652: 119 BPL RDATA4

```

```

11 BOOT          Disk II boot code
C600:59 D6 02      EOR    DNIBL-$80,Y
C603:D0 CD C6A2    120    BNE    BADRD01
C605:A0 00         121    LDY    #500
C607:A2 56         122    LDY    #556
C609:CA          123    DENIBL
C60A:30 FB C607    124    DENIB1
C60C:B1 26         125    BML    DENIBL
C60E:5E 00 03     126    LDA    ($26),Y
C612:2A          127    LSR    NBUF1,X
C615:5E 00 03     128    ROL    A
C618:2A          129    LSR    NBUF1,X
C61A:5E 00 03     130    ROL    A
C61D:2A          131    STA    ($26),Y
C61F:5E 00 03     132    INY    DENIB1
C622:2A          133    BNE    DENIB1
C625:2A          134    *****
C628:C8          135    * Code beyond this point is not *
C62A:5E 00 03     136    * sacred... It may be perverted *
C62D:5E 00 03     137    * in any manner by any pervert... *
C630:5E 00 03     138    *****
C633:5E 00 03     139    INY    INC $27
C636:5E 00 03     140    INC $30
C639:5E 00 03     141    LDA $30
C63C:5E 00 03     142    CMP $0800
C63F:5E 00 03     143    LDY    BOOTDEV
C642:5E 00 03     144    BCC    BADRD01
C645:5E 00 03     145    JMP $0801
C648:5E 00 03     146    DS    $C700-4,0
C64B:5E 00 03     147    include mouse
C64E:5E 00 03     148    ;Last byte must be 0
C651:5E 00 03     149    ;Equates for the mouse.
C654:5E 00 03     150
C657:5E 00 03     151
C65A:5E 00 03     152
C65D:5E 00 03     153
C660:5E 00 03     154
C663:5E 00 03     155
C666:5E 00 03     156
C669:5E 00 03     157
C66C:5E 00 03     158
C66F:5E 00 03     159
C672:5E 00 03     160
C675:5E 00 03     161
C678:5E 00 03     162
C67B:5E 00 03     163
C67E:5E 00 03     164
C681:5E 00 03     165
C684:5E 00 03     166
C687:5E 00 03     167
C68A:5E 00 03     168
C68D:5E 00 03     169
C690:5E 00 03     170
C693:5E 00 03     171
C696:5E 00 03     172
C699:5E 00 03     173
C69C:5E 00 03     174
C69F:5E 00 03     175
C6A2:5E 00 03     176
C6A5:5E 00 03     177
C6A8:5E 00 03     178
C6AB:5E 00 03     179
C6AE:5E 00 03     180
C6B1:5E 00 03     181
C6B4:5E 00 03     182
C6B7:5E 00 03     183
C6BA:5E 00 03     184
C6BD:5E 00 03     185
C6C0:5E 00 03     186
C6C3:5E 00 03     187
C6C6:5E 00 03     188
C6C9:5E 00 03     189
C6CC:5E 00 03     190
C6CF:5E 00 03     191
C6D2:5E 00 03     192
C6D5:5E 00 03     193
C6D8:5E 00 03     194
C6DB:5E 00 03     195
C6DE:5E 00 03     196
C6E1:5E 00 03     197
C6E4:5E 00 03     198
C6E7:5E 00 03     199
C6EA:5E 00 03     200
C6ED:5E 00 03     201
C6F0:5E 00 03     202
C6F3:5E 00 03     203
C6F6:5E 00 03     204
C6F9:5E 00 03     205
C6FC:5E 00 03     206
C700:5E 00 03     207

```

```

20-OCT-86 06:41 PAGE 32

Mouse firmware

12 MOUSE
C700: 2          msb ON
C700: 3 *****
C700: 4 *
C700: 5 * Mouse firmware for the Chels
C700: 6 *
C700: 7 * by Rich Williams
C700: 8 * July, 1983
C700: 9 *
C700: 10 *****
C700: 11 *****
C700: 12 *****
C700: 13 *
C700: 14 * Equates
C700: 15 *
C700: 16 *****
C700: 17 *****
C700: 18 * Input bounds are in scratch area
C700: 19 mountemp equ $478 ;temporary storage
C700: 20 minl equ $478
C700: 21 maxl equ $4F8
C700: 22 minh equ $578
C700: 23 maxh equ $5F8
C700: 24 * Mouse bounds in slot 5 screen area
C700: 25 minxl equ $47D
C700: 26 minyl equ $4FD
C700: 27 minxh equ $57D
C700: 28 minyh equ $5FD
C700: 29 maxxl equ $67D
C700: 30 maxyl equ $6FD
C700: 31 maxxh equ $77D
C700: 32 maxyh equ $7FD
C700: 33 * Mouse holes in slot 7 screen area
C700: 34 mouxl equ $47F ;X position low byte
C700: 35 mouyl equ $4FF ;Y position low byte
C700: 36 mouxh equ $57F ;X position high byte
C700: 37 mouyh equ $5FF ;Y position high byte
C700: 38 mouarm equ $67F ;Arm interrupts from movement or button
C700: 39 moustat equ $77F ;Mouse status
C700: 40 * Moustat provides the following
C700: 41 * D7= Button pressed
C700: 42 * D6= Status of button on last read
C700: 43 * D5= Moved since last read
C700: 44 * D4= Reserved
C700: 45 * D3= Interrupt from VBL
C700: 46 * D2= Interrupt from button
C700: 47 * D1= Interrupt from movement
C700: 48 * D0= Reserved
C700: 49 moumode equ $7FF ;Mouse mode
C700: 50 * D7 = 1 if user wants control of mouse interrupts
C700: 51 * D6-D4= Unused
C700: 52 * D3= VBL active
C700: 53 * D2= VBL interrupt on button
C700: 54 * D1= VBL interrupt on movement
C700: 55 * D0= Mouse active

```

```

C700: 0020 56 movarm equ $20
C700: 000C 57 vblmode equ $0C
C700: 0004 58 butmode equ $04
C700: 0002 59 movmode equ $02

61 * Hardware addresses
C015 62 mouxint equ $C015
C017 63 mouyint equ $C017
C019 64 vblint equ $C019
C078 65 loudshl equ $C078
C079 66 loudshl equ $C079
C048 67 mouclr equ $C048
C058 68 ion equ $C058
C059 69 moudshl equ $C059
C063 70 moudshl equ $C063
C066 71 moudshl equ $C066
C067 72 mouxl equ $C067
C070 73 vblclr equ $C070
C700: 75 *
C700: 76 * Other addresses
C700: 77 *
C020 78 lnbuff equ $200
C0214 79 bial equ inbuf+20
C0215 80 binh equ inbuf+21
C700: 50 include mcode.x

;D7 = x interrupt
;D7 = y interrupt
;D7 = vbl interrupt
;Disable iou access
;Clear mouse interrupt
;IOU interrupt switches
;Disable mouse interrupts
;Disable mouse interrupts
;D7 = Mouse button
;D7 = Y1
;Clear VBL interrupt

;Input buffer
;Temp for binary conversion

C700: 2 *****
C700: 3 *
C700: 4 * Entry points for mouse firmware
C700: 5 *
C700: 6 *****
C700: 7 mbasic bra outent
C702:82 03 8 pnull idx #3
C704:60 9 rts
C705:38 10 inent sec
C706:90 11 dfb $90
C707:18 12 outent clic
C708:4C CF 05 13 jmp xmbasic
C709:01 14 dfb $01
C70A:20 15 dfb $20
C70B:02 16 dfb $null
C70C:02 17 dfb $null
C70D:02 18 dfb $null
C70E:02 19 dfb $null
C70F:00 20 dfb $0
C710:00 21 dfb $xsetmou
C711:00 22 dfb $xmtstint
C712:1C 23 dfb $xread
C713:22 24 dfb $xmclear
C714:28 25 dfb $xnoerror
C715:2E 26 dfb $xclamp
C716:1A 27 dfb $xhome
C717:34 28 dfb $initmouse
C718:3A 29 dfb $null
C719:40 30 dfb $goinit
C71A:
C71A:18 32 noerror clic
C71B:60 33 rts
C71C:8D 28 C0 35 xsetmou sta rombank
C71F:4C C2 C6 36 jmp sw.setmou
C722:8D 28 C0 38 xmtstint sta rombank
C725:4C CD C6 39 jmp sw.mtstint
C728:8D 28 C0 41 xread sta rombank
C72B:4C D8 C6 42 jmp sw.mread
C72E:8D 28 C0 44 xmclear sta rombank
C731:4C E3 C6 45 jmp sw.mclear
C734:8D 28 C0 47 xclamp sta rombank
C737:4C EE C6 48 jmp sw.mclamp
C73A:8D 28 C0 50 xhome sta rombank
C73D:4C F9 C6 51 jmp sw.mhome
C740:8D 28 C0 53 initmouse sta rombank
C743:4C 04 C7 54 jmp sw.initmouse
C746:8D 28 C0 56 sta rombank
C749:4C 9A CF 57 jmp m.oveirq
C74C:8D 28 C0 59 slboot sta rombank

```

13 MODE.X

Mouse firmware

20-OCT-86 06:41 PAGE 35

14 SWITCHER

Mouse firmware

20-OCT-86 06:41 PAGE 36

```

C74F:4C B4 C6      jmp    swsl.bt      ;other side
C752:8D 28 C0      sta    rombank
C753:DA             ptx
C756:20 16 C8      jsr    getic
C759:5A             phy
C75A:8C 78 06      sty    slcstate
C75D:20 00 D8      jsr    execute
C760:4C 0E C8      jmp    fixic
C763:             ds      $C780-*, $00
C780:             include switcher ;Bank switcher @ $C780

```

```

C780:             2      ds      $C780-*, 0
C780:             3      *
C780:             4      *
C780:             5      * Code for switching between banks
C780:             6      * This code appears in both banks of the rom
C780:             7      *
C780:             8      *
C780:             9      swrti sta rombank ;RTI to the other bank
C783:40           10     rti
C784:8D 28 C0     11     swrts sta rombank ;RTS to the other bank
C787:60           12     swrts rts
C788:8D 28 C0     13     swreset sta rombank ;Reset routine
C789:4C 62 FA     14     jmp reset
C78B:8D 28 C0     15     sta rombank ;Interrupt routine
C791:2C 87 C7     16     bit swrtsop ;Set V = 1 for other bank
C794:4C 04 C8     17     jmp lrgnt
C797:8D 28 C0     18     swpmv sta rombank ;Protocol converter
C79A:4C F1 C7     19     jmp swsthk3 ;Jump to sethooks from other side
C79B:8D 28 C0     20     swbasicin sta rombank ;Mouse BASIC routines
C7A0:4C F6 C7     21     jmp swzqt3 ;Jump to zquit from other side
C7A3:8D 28 C0     22     swttm sta rombank ;Set terminal mode
C7A6:4C F1 C7     23     jmp swstm3 ;Serial port command processor
C7A9:8D 28 C0     24     swcmd sta rombank ;Moveaux
C7AC:4C 06 C8     25     jmp swcmd3
C7AF:8D 28 C0     26     swaux sta rombank
C7B2:4C 4E C3     27     jmp moveaux
C7B5:8D 28 C0     28     swxfer sta rombank
C7B8:4C 97 C3     29     jmp xfer
C7BB:8D 28 C0     30     swmint sta rombank ;Mouse Interrupt handler
C7BE:4C 00 C1     31     jmp mouseint
C7C1:8D 28 C0     32     banger sta rombank
C7C4:4C 8E D4     33     jmp diags
C7C7:8D 28 C0     34     swatalk sta rombank ;Jump to appletalk
C7CA:4C 80 C5     35     jmp atalk
C7CD:8D 28 C0     36     swser3 sta rombank ;Jump to serout3
C7D0:4C 4F C2     37     jmp serout3
C7D3:8D 28 C0     38     swgetst sta rombank ;Jump to getstat
C7D6:4C AC C2     39     jmp getstat
C7D9:8D 28 C0     40     swread sta rombank
C7DC:4C C3 C2     41     jmp xrdser ;Jump to xrdser
C7DF:8D 28 C0     42     swgetb sta rombank ;Jump to getbuf
C7E2:4C F7 C2     43     jmp getbuf
C7E5:8D 28 C0     44     swzznm sta rombank
C7E8:4C C5 D4     45     jmp zznm
C7EB:8D 28 C0     46     swxfgo sta rombank ;Jump to users xfer dest
C7EE:4C ED 03     47     jmp ($3ED)
C7F1:20 23 CE     48     swsthk3 jsr sethooks
C7F4:80 8E C784   49     bra swrts
C7F6:20 4D CE     50     swzqt3 jsr zquit
C7F9:80 89 C784   51     bra swrts
C7FB:D6           53     dfb $D6 ;should be at $C7FB
C7FC:             55     ds $C7FF-*, 0
C7FF:00           56     dfb 0 ;Appletalk version number
C800:             52     include irqbuf ;Interrupt stuff @ $C800

```

```

C800: 3 *****
C801: 4 *
C802: 5 * NEWIRQ - The main (only) IRQ handling routines
C803: 6 * IRQENT - Entry point from alternate rom bank
C804: 7 *
C805: 8 *
C806: 9 * This routine saves the memory state of the machine,
C807: 10 * checks for an internal interrupt, and then calls the user's
C808: 11 * interrupt handler at $3FE.
C809: 12 * The memory state is encoded as follows:
C810: 13 * D7 = 1 if Alternate zero page / stack
C811: 14 * D6 = 1 if 80 store and page 2
C812: 15 * D5 = 1 if Read aux
C813: 16 * D4 = 1 if Write Aux
C814: 17 * D3 = 1 if L.C. enabled
C815: 18 * D2 = 1 if L.C. and $D000 bank 1
C816: 19 * D1 = 1 if L.C. and $D000 bank 2
C817: 20 * D0 = 1 if Alternate rom bank
C818: 21 *
C819: 22 * New changes in the interrupt handler are marked with a +
C820: 23 *
C821: 24 *****
C822: 25      jmp plinit      ;Pascal 1.0 Initialization
C823: 26 NEWIRQ EQU *      ;+
C824: 27 IRQENT EQU *      ;+ V=0 for main bank
C825: 28      pha            ;+ Entry pt from other bank assumes V=1
C826: 29      pex            ;+ Save A on stack, not $45
C827: 30      tsx            ;+ X too
C828: 31      pla            ;+ Save stack pointer
C829: 32      pla            ;+ Skip past X
C830: 33      pla            ;+ And A
C831: 34      pha            ;+ Here is the status Oh boy!
C832: 35      txs            ;+ Fix the stack pointer
C833: 36      phy            ;+ Save Y too
C834: 37      ldx            ;+ Get mouse info
C835: 38      ldy            ;+ As soon as we can
C836: 39      cld            ;+ No decimal mode please
C837: 40      and $10         ;+ Test break bit
C838: 41      cmp $50         ;+ C=1 if break, V unchanged
C839: 42      lda r08col      ;+ TEST FOR 80-STORE WITH
C840: 43      and rdpag2      ;+ PAGE 2 TEXT
C841: 44      and $80         ;+ MAKE IT ZERO OR $80
C842: 45      beq irq2        ;+
C843: 46      sta txtpage1    ;+ SET PAGE 2 RESPT BIT.
C844: 47      lda $540        ;+ Which Rombank?
C845: 48 irq2      bvc irq21   ;+ Mark other bank
C846: 49      ror            ;+
C847: 50 irq21     brr rdmrwd  ;+
C848: 51      bpl irq3        ;+ BRANCH IF MAIN RAM READ
C849: 52      stl rmainram    ;+ ELSE, SWITCH IT IN
C850: 53      ora $520        ;+ AND RECORD THE EVENT!
C851: 54 irq3      bit rdmrwt  ;+ DO THE SAME FOR RAM WRITE.
C852: 55      bpl irq4        ;+
C853: 56      sta rmainram   ;+
C854: 57      ora $510        ;+
C855: 58 irq4      bcs irq5   ;+ Branch if break
C856: 59      pha            ;+ Save machine states so far....
C857: 60      jsr swint       ;+ Go Test Mouse & ACIA

```

```

C858: 61      irq5col         ;+ Branch if it was. IC unchanged!
C859: 62      pia            ;+ Restore states recorded so far
C860: 63      cic            ;+ Reset break/interrupt handler
C861: 64 irq5      bit r08col  ;+ DETERMINE IF LANGUAGE CARD ACTIVE
C862: 65      bra passkip1    ;+ Skip around pascal 1.0 stuff
C863: 66      ds $C84D-$, $00
C864: 67      jmp pload      ;+
C865: 68 passkip1 equ *      ;+
C866: 69      bpl irq7        ;+ SET TWO BITS SO RESTORED
C867: 70      ora $FC         ;+ LANGUAGE CARD IS WRITE ENABLED
C868: 71      bit r08cnk2     ;+ BRANCH IF NOT PAGE 2 OF $D000
C869: 72      bpl irq6        ;+ ENABLE READ FOR PAGE 2 ON EXIT
C870: 73      eor $56        ;+
C871: 74 irq6      sta r08min  ;+ LAST...AND VERY IMPORTANT!
C872: 75 irq7      bit rdmaltzp ;+ UNLESS IT IS NOT ENABLED
C873: 76      bpl irq8        ;+ SAVE CURRENT STACK POINTER
C874: 77      tsx            ;+ AT BOTTOM OF STACK
C875: 78      stx $101        ;+ GET MAIN STACK POINTER
C876: 79      ldx $100        ;+
C877: 80      txs            ;+
C878: 81      sta setstozp   ;+
C879: 82      ora $580        ;+
C880: 83 irq8      bcs gobreak  ;+
C881: 84      pha            ;+
C882: 85      lda $<IRODNE    ;+
C883: 86      pha            ;+
C884: 87      lda $>IRODNE    ;+ SAVE RETURN IRQ ADDR
C885: 88      pha            ;+
C886: 89      lda $4         ;+ SO WHEN INTERRUPT DOES RTI
C887: 90      pha            ;+ IT RETURNS TO IRODNE.
C888: 91      jmp ($3FE)      ;+ PROCESS EXTERNAL INTERRUPT
C889:
C890: 93 * The user's RTI returns here
C891: 94 * REMAKE
C892: 95 * The rom must be reenabled with a LDA romin
C893: 96 * This way if the IC was write protected, it still is
C894: 97 * If it was write enabled, it still is
C895: 98 * If it was being write enabled ( 2 lds), it still will be
C896: 99 * The restore loop uses an INC because some of the switches are read
C897: 100 * and some are write. It must be an INC abs,x since both the 6502 and
C898: 101 * the 65C02 do two reads before the write (for different reasons).
C899: 102 iroqone     lda romin  ;+ Did some clown bank out the rom?
C900: 103 irq5col     pla        ;+ Recover machine state
C901: 104      bpl irqm1        ;+ Branch if main zp was active
C902: 105      sta setaltzp    ;+ Restore alternate stack pointer
C903: 106      ldx $101        ;+
C904: 107      txs            ;+
C905: 108 irqm1     ldy $906    ;+ Y = index into table of switches
C906: 109 irqm2     bpl irqm3    ;+ Branch if no change
C907: 110      ldx irqtable,y    ;+ Get soft switch address
C908: 111      inc $C000,X      ;+ Hit the switch. No page cross!!!
C909: 112 irqm3     dey        ;+
C910: 113      bmi irqm4       ;+ Branch if all done
C911: 114      asl A           ;+ Get next bit to check
C912: 115      bne irqm2       ;+ Fall through if all done
C913: 116 irqm4     asl A       ;+ C = 1 if other rom bank
C914: 117      asl A           ;+

```

## Keyboard buffering

15 INQBUF

```

C8C0:      162 * The following routine is for reading key-
C8C1:      163 * board from buffers or directly.
C8C2:      164 * Type-ahead buffering only occurs for non auto-
C8C3:      165 * repeat keypresses. When a key is pressed for
C8C4:      166 * auto-repeat the buffer is first emptied, then the
C8C5:      167 * repeated characters are returned.
C8C6:      168 * The minus flag is used to indicate if a keystroke
C8C7:      169 * is being returned.
C8C8:      170 *

```

```

C8C9:      172 XKBED1 LDA KBD      ;test keyboard directly
C8CA:      173 BPL XKBED      ;loop if buffered since test.
C8CB:      174 STA KBDSTRB      ;Clear keyboard strobe.
C8CC:      175 XMOKEY RTS        ;Minus flag indicates valid character
C8CD:      177 XKBED JSR XBITKBD   ;is keyboard input ready?
C8CE:      178 BPL XMOKEY      ;Branch if not.
C8CF:      179 BCC XKBED1      ;Branch if direct KBD input.
C8D0:      180 PHN            ;Save Y
C8D1:      181 LDY #80          ;Y=80 for keyboard buffer
C8D2:      182 JSR SW65TB      ;Get data from buffer
C8D3:      183 PLY            ;Set minus flag
C8D4:      184 ORA #0
C8D5:      185 RTS

```

```

C8D6:      187 XBITKBD BIT TYPED   ;This routine replaces "BIT KBD"
C8D7:      188 BPL XKBED2      ;Instructions so as to function with
C8D8:      189 ;               type-ahead
C8D9:      190 SEC              ;anticipate data in buffer is ready
C8DA:      191 PHP            ;save carry and minus flags
C8DB:      192 PEA            ;preserve accumulator
C8DC:      193 LDA TKEY        ;is there data to be read?
C8DD:      194 CMP TKEY        ;branch if type-ahead buffer empty
C8DE:      195 BEQ XKBED1
C8DF:      196 PLA
C8E0:      197 PIP
C8E1:      198 RTS
C8E2:      199 *
C8E3:      200 XKBED1 PLA
C8E4:      201 PIP
C8E5:      202 XKBED2 BIT KBD      ;restore ACC and Status
C8E6:      203 CLC              ;test KBD Directly
C8E7:      204 RTS              ;indicate direct test

```

```

C900:      206 *****
C901:      207 *
C902:      208 * PADDLE patch
C903:      209 * This routine returns the mouse position instead of
C904:      210 * the paddle if the mouse is on
C905:      211 *
C906:      212 *****
C907:      213 mpaddle equ *
C908:      214 lda mousemode      ;Is the mouse active?
C909:      215 cmp #01              ;Only transparent mode
C90A:      216 beq pdon
C90B:      217 lda vblclr          ;Fire the strobe

```

## Serial &amp; Keyboard buffering

15 INQBUF

```

C892:      118 PLY
C893:      119 PLY
C894:      120 PLA
C895:      121 BCS INQ5
C896:      122 RTI
C897:      123 INQ5 JMP SWRTI
C898:      124 * Go back to the other bank
C899:      125 *****
C89A:      126 *
C89B:      127 * GDBREAK- If a break instruction has occurred, we check
C89C:      128 * if the BRK happened in the alternate rom bank. If it has,
C89D:      129 * some fool may have hit the rom switch by accident and the PC is
C89E:      130 * decremented by two, the main rom is switched in and we resume
C89F:      131 * where we think he wanted to go
C8A0:      132 *
C8A1:      133 *****
C8A2:      134 GDBREAK EQU *
C8A3:      135 BMI GBRBK
C8A4:      136 BIT #9
C8A5:      137 BEQ GBRBK
C8A6:      138 AND #7FE
C8A7:      139 PHA
C8A8:      140 TSX
C8A9:      141 PLA
C8AA:      142 PLA
C8AB:      143 PLA
C8AC:      144 PLA
C8AD:      145 PLA
C8AE:      146 PLA
C8AF:      147 PLY
C8B0:      148 CPT #SCI
C8B1:      149 BCC GBNOTROM
C8B2:      150 SEC #2
C8B3:      151 BCS GBNOC
C8B4:      152 DFF
C8B5:      153 GBNOC
C8B6:      154 PHN
C8B7:      155 TYS
C8B8:      156 JMP INQ5
C8B9:      157 GBNOTROM TYS
C8BA:      158 PLA
C8BB:      159 GBRBK JMP NEWBRK

```

```

C8A7:      134 GDBREAK EQU *
C8A8:      135 BMI GBRBK
C8A9:      136 BIT #9
C8AA:      137 BEQ GBRBK
C8AB:      138 AND #7FE
C8AC:      139 PHA
C8AD:      140 TSX
C8AE:      141 PLA
C8AF:      142 PLA
C8B0:      143 PLA
C8B1:      144 PLA
C8B2:      145 PLA
C8B3:      146 PLA
C8B4:      147 PLY
C8B5:      148 CPT #SCI
C8B6:      149 BCC GBNOTROM
C8B7:      150 SEC #2
C8B8:      151 BCS GBNOC
C8B9:      152 DFF
C8BA:      153 GBNOC
C8BB:      154 PHN
C8BC:      155 TYS
C8BD:      156 JMP INQ5
C8BE:      157 GBNOTROM TYS
C8BF:      158 PLA
C8C0:      159 GBRBK JMP NEWBRK

```

Keyboard buffering

```

C90A:4C 21 F8 C90D      jmp     $F821
C90D:      equ     +
C90D:E0 01 C90D      cpx     #1
C90F:6A      ror     A
C910:A8      tay
C911:89 7F 05 C918      lda     mouth,y
C914:F0 02 C918      beq     pdok
C916:A9 FF      lda     $FF
C918:19 7F 04 C918      ora     mouth,y
C91B:A8      tay
C91C:60      rts
C91D:      include mini

```

;Mini assembler & step routines

```

3 *****
4 *
5 * Apple //c Mini Assembler
6 *
7 * Got mnemonic, check address mode
8 *
9 *****
10 AMOD1 JSR WBL      ;get next non-blank
11      STY YSAV      ;save Y
12      CMP CHAR1,X
13      BNE AMOD2
14      JSR WBL
15      CMP CHAR2,X
16      BEQ AMOD3
17      LDA CHAR2,X
18      BEQ AMOD4
19      CMP #5AA
20      BEQ AMOD4
21      LDY YSAV
22      CLC
23      DEX
24      ROL
25      CFX
26      BNE AMOD6
27      JSR GETNUM
28      LDA A2H
29      BEQ AMOD5
30      INX
31      STX YSAV1
32      LDX #503
33      DEY
34      STX A1H
35      DEX
36      BPL AMOD1
37      RTS

39 *
40 *
41 * Calculate offset byte for relative addresses
42 *
43 REL      SRC #501      ;calc relative address
44      LSR A
45      BNE GOERR
46      LOY A2H
47      LDX A2L
48      BNE REL1
49      DEY
50      REL1      DEX
51      TXA
52      CLC
53      SRC PCL
54      STA A2L
55      BPL REL2
56      INY
57      REL2      TYA
58      SRC PCE

```

16 MINI	65C02 Mini assembler	20-OCT-86 06:41 PAGE 43
C98E:D0 57	C9C7	59 GOERR BNE MINERR
C970:		60 * ;display error
C970:		61 * Move instruction to memory
C970:		62 * ;get instruction length
C970:A4 2F		63 MOVINST LDA A1B.Y
C972:B9 3D 00		64 MOV1 STA (PCL),Y
C975:F1 3A		65 ;and move it
C977:88		66 DEY
C978:10 F8	C972	67 BPL MOV1
C97A:		68 * Display instruction
C97A:		70 * ;print blanks to make PRODOS work
C97A:20 48 F9		71 JSR PRBLNK
C97D:20 1A FC		72 JSR UP
C980:20 1A FC		73 JSR UP
C983:	C983	74 DISLIN EQU *
C983:20 C4 C5		75 JSR SHOWINST
C986:	C986	76 GETINST1 EQU *
C986:A9 A1		77 LDA \$9A1
C988:85 33		78 STA PROMPT
C98A:20 67 FD		79 JSR GETINST2
C98D:80 49	C9D8	80 BRA DOINST
C98F:		81 * Compare disassembly of all known opcodes with
C98F:		82 * the one typed in until a match is found
C98F:		84 * ;get opcode
C98F:A5 3D		85 GETOP LDA A1B
C991:20 8E F8		86 JSR INSDS2
C994:AA		87 TAX
C995:BD 00 FA		88 LDA MNEMR,X
C996:C5 42		89 CMP A4L
C99A:D0 21	C9BD	90 BNE NXTOP
C99C:BD C0 F9		91 LDA MNEML,X
C99F:80 0C	C9AD	93 bra plskip
C9A1:	0009	94 ds \$C9A1-*,0
C9AA:4C B4 C1		95 jmp plwrite
C9AD:	96 plskip	
C9AD:C5 43		98 CMP A4B
C9AF:D0 0C	C9BD	99 BNE NXTOP
C9B1:A5 44		100 LDA A5L
C9B3:A4 2E		101 LDY FORMAT
C9B5:C0 9D		102 CPY \$9D
C9B7:F0 9C	C955	103 BEQ REL
C9B9:C5 2E		104 CMP FORMAT
C9BD:F0 B3	C970	105 BEQ MOVINST
C9BD:C6 3D		106 DEC A1B
C9BF:D0 CE	C98F	107 BNE GETOP
C9C1:E6 44		108 INC A5L
C9C3:C6 35		109 DEC YSAV1
C9C5:F0 C8	C98F	110 BEQ GETOP
C9C7:	111 *	
C9C7:		112 * Point to the error with a caret, beep, and fall
C9C7:		113 * into the mini-assembler.
C9C7:		114 *
C9C7:A4 34		115 MINERR LDY YSAV
C9C9:98		116 ERR2 TYA

16 MINI	65C02 Mini assembler	20-OCT-86 06:41 PAGE 44
C9CA:AA	117	TAX
C9CB:	C9CB	118 ERR3 EQU *
C9C8:20 4A F9		119 JSR PRBL2
C9CE:A9 DE		120 LDA \$DE
C9D0:20 0D FD		121 JSR COUT
C9D3:20 3A FF		122 JSR BELL
C9D6:80 AE	C986	123 BRA GETINST1
C9D8:		124 *
C9D8:		125 * Read a line of input. If prefaced with " ", decode
C9D8:		126 * mnemonic. If "\$" do monitor command. Otherwise parse
C9D8:		127 * hex address before decoding mnemonic.
C9D8:		128 *
C9D8:20 C7 FF		129 DOINST JSR ZMODE
C9D8:AD 00 02		130 LDA \$200
C9DE:C9 AD		131 CMP \$9AD
C9E0:F0 12	C9F4	132 BEQ DOLIN
C9E2:C9 8D		133 CMP \$98D
C9E4:D0 01	C9E7	134 BNE GETI1
C9E6:60		135 RTS
C9E7:	136 *	
C9E7:20 A7 FF		137 GETI1 JSR GETNUM
C9EA:C9 93		138 CMP \$993
C9EC:D0 DB	C9C9	139 GOERR2 BNE ERR2
C9EE:8A		140 TYA
C9EF:F0 D8	C9C9	141 BEQ ERR2
C9F1:	142 *	
C9F1:20 78 FE		143 JSR A1PCPL
C9F4:A9 03		144 LDA \$903
C9F6:85 3D		145 STA A1B
C9F8:20 3B CA		146 NTCH JSR NBL
C9FB:0A		147 ASL A
C9FC:E9 BE		148 SBC \$BE
C9FE:C9 C2		149 CMP \$C2
C9A0:90 C7	C9C9	150 BCC ERR2
C9A2:	151 *	
C9A2:		152 * Form mnemonic for later comparison
C9A2:	153 *	
C9A2:0A		154 ASL A
C9A3:0A		155 ASL A
C9A4:A2 04		156 LDX \$904
C9A6:0A		157 NYTMN ASL A
C9A7:26 42		158 ROL A4L
C9A9:26 43		159 ROL A4H
C9AB:CA		160 DEX
C9AC:10 F8	CA06	161 BPL NYTMN
C9AD:C6 3D		162 DEC A1B
C9AF:F0 F4	CA06	163 BEQ NYTMN
C9B2:10 E4	C9F8	164 BPL NTCH
C9B4:A2 05		165 LDX \$5
C9B6:20 1D C9		166 JSR AMOD1
C9B9:A5 44		167 LDA A5L
C9BB:0A		168 ASL A
C9BC:0A		169 ASL A
C9BD:05 35		170 ORA YSAV1
C9BF:C9 20		171 CMP \$920
C9C1:B0 06	CA29	172 BCS AMOD7
C9C3:A6 35		173 LDX YSAV1
C9C5:F0 02	CA29	174 BEQ AMOD7

421



```

CB2C: 0004      3 ;align this for fools with illegal entry points
CB2C: 4         ds $CB30-*,0
CB30: 5         6 * SCROLLIT scrolls the screen either up or down, depending
CB30: 6         7 * on the value of X. It scrolls within windows with even
CB30: 7         8 * or odd edges for both 40 and 80 columns. It can scroll
CB30: 8         9 * windows down to 1 character wide.
CB30: 9         10 *
CB30: 10        11 SCROLLDN PEX ;save X
CB30: 11        12 LDX #0 ;direction = down
CB31: 12        13 BRA SCROLLIT ;do scroll
CB33: 13        14 *
CB35: 14        15 SCROLLUP PEX ;save X
CB35: 15        16 LDX #1 ;direction = up
CB36: 16        17 SCROLLIT LDX WNDWTH ;get width of screen window
CB38: 18        18 BIT R080VID ;in 40 or 80 columns?
CB3A: 19        19 BPL GETST ;=>40, determine starting line
CB3D: 20        20 STA SETWCOL ;make sure this is enabled
CB3F: 21        21 TTA A ;get WNDWTH for test
CB41: 22        22 LSR A ;divide by 2 for 80 column index
CB44: 23        23 TAY ;and save
CB45: 24        24 LDA A ;test oddity of right edge
CB47: 25        25 LSR A ;by rotating low bit into carry
CB48: 26        26 CLV ;V=0 if left edge even
CB49: 27        27 BCC CHKPT ;=>check right edge
CB4B: 28        28 BIT SEVI ;V=1 if left edge odd
CB4E: 29        29 CHKPT ;restore WNDWTH
CB4F: 30        30 EOR A ;get oddity of right edge
CB51: 31        31 LSR A ;C=1 if right edge even
CB52: 32        32 BVS GETST ;if odd left, don't DEY
CB54: 33        33 BCS GETST ;if odd right, don't DEY
CB56: 34        34 DEY ;if even right edge odd, need one less
CB57: 35        35 GETST ;save window width
CB5A: 36        36 LDA R080VID ;N=1 if 80 columns
CB5D: 37        37 PIP ;save N,2,V
CB5E: 38        38 LDA WNDTOP ;assume scroll from top
CB60: 39        39 CFX #0 ;up or down?
CB62: 40        40 BNE SETDBAS ;->up
CB64: 41        41 LDA WNDPTH ;down, start scrolling at bottom
CB66: 42        42 DEC A ;really need one less
CB67: 43        43 *
CB67: 44        44 SETDBAS STA TMPA ;save current line
CB6A: 45        45 JSR VTBAS ;calculate base with window width
CB6D: 46        46 *
CB6D: 47        47 SCRLIN LDA BASL ;current line is destination
CB6E: 48        48 STA BASL ;
CB6F: 49        49 LDA BASH ;
CB71: 50        50 STA BAS2H ;
CB75: 51        51 *
CB75: 52        52 LDA TMPA ;get current line
CB78: 53        53 CFX #0 ;going up?
CB7A: 54        54 BNE SETUP2 ;->up, inc current line
CB7C: 55        55 CPE WNDTOP ;down. Reached top yet?
CB7E: 56        56 BEQ SCRL3 ;yes! clear top line, exit
CB80: 57        57 DEC A ;no, go up a line
CB81: 58        58 BRA SETSRC ;set source for scroll
CB83: 59        59 SETUP2 ;up, inc current line
CB84: 60        60 INC WNDPTH ;at bottom yet?

```

```

CB86: 61        61 CB89 BCS SCRL3 ;yes! clear bottom line, exit
CB88: 62        62 *
CB88: 63        63 SETSRC STA TMPA ;save new current line
CB8B: 64        64 JSR VTBAS ;get base for new current line
CB8E: 65        65 LDA WNDPTH ;get width for scroll
CB91: 66        66 PIP ;get status for scroll
CB92: 67        67 PIP ;N=1 if 80 columns
CB93: 68        68 BPL SKPT ;->only do 40 columns
CB95: 69        69 LDA TMPAGE2 ;scroll aux page first (even bytes)
CB98: 70        70 TTA ;test Y
CB99: 71        71 BEQ SCRL3 ;if Y=0, only scroll one byte
CB9B: 72        72 SCRLIN LDA (BASL),Y
CB9D: 73        73 STA (BAS2L),Y
CB9F: 74        74 DEY ;
CBAA: 75        75 BNE SCRLIN ;do all but last even byte
CBAB: 76        76 SCRLIN BVS SKPT ;odd left edge, skip this byte
CBAD: 77        77 LDA (BASL),Y ;
CBAE: 78        78 STA (BAS2L),Y ;
CBAD: 79        79 SKPT ;now do main page (odd bytes)
CBAB: 80        80 LDA WNDPTH ;restore width
CBAB: 81        81 BCS SKPT ;even right edge, skip this byte
CBAB: 82        82 SCRLIN LDA (BASL),Y ;
CBAB: 83        83 STA (BAS2L),Y ;
CBAB: 84        84 SKPT ;
CBAB: 85        85 BPL SCRLIN ;scroll next line
CBAB: 86        86 BRA SCRLIN ;
CBAB: 87        87 *
CBAB: 88        88 SCRLIN JSR VTBAS ;clear current line
CBAB: 89        89 JSR VTBAS ;restore original cursor line
CBAB: 90        90 PIP ;pull status off stack
CBAB: 91        91 PIP ;restore X
CBAB: 92        92 SEVI ;done!!!

```

```

C8C2: 94 * DOCLR is called by CLREOM. It decides whether
C8C2: 96 * to do a (quick) 40 or 80 column clear to end of line.
C8C2: 97 *
C8C2: 98 DOCLR BIT R08VID ;40 or 80 column clear?
C8C5: 99 BMT CLR80 ;(RASI),Y
C8C7: 99 STA CLR40
C8C9: 101 INY ;=>clear 80 columns
C8CA: 101 INY
C8CB: 102 CPY RNDMOTH
C8CC: 102 BCC CLR40
C8CD: 103 RTS
C8CE: 104
C8CF: 105 *
C8D0: 106 CLREALF
C8D1: 107 LDX #908
C8D2: 108 LDY #20
C8D3: 109 LDA INVTIG
C8D4: 109 LDA INVTIG
C8D5: 110 AND #9A0
C8D6: 110 BKA CLR2
C8D7: 111 *
C8D8: 112 *
C8D9: 113 PEX
C8DA: 114 PBA
C8DB: 114 PBA
C8DC: 115 TIA
C8DD: 116 SEC
C8DE: 117 SEC
C8DF: 118 SEC
C8E0: 119 TAY
C8E1: 120 TIA
C8E2: 121 ASR
C8E3: 122 TAY
C8E4: 123 PBA
C8E5: 123 PBA
C8E6: 124 XOR
C8E7: 125 XOR
C8E8: 126 BCS
C8E9: 127 BCS
C8EA: 128 INY
C8EB: 129 CLR0
C8EC: 130 PBA
C8ED: 131 CLR2
C8EE: 132 STA
C8EF: 133 BIT
C8F0: 134 INX
C8F1: 135 BEQ
C8F2: 136 CLR3
C8F3: 137 INY
C8F4: 138 INX
C8F5: 139 BNE
C8F6: 140 CLR3
C8F7: 141 RTS
C8F8: 142 *
C8F9: 143 CLRPORT
C8FA: 144 STZ
C8FB: 144 STZ
C8FC: 145 RTS

```

```

C8B8: 147 *
C8B9: 148 * PASTINVERT is used by Pascal to display the cursor. Pascal
C8BA: 149 * normally leaves the cursor on the screen at all times. It
C8BB: 150 * is fleetingly removed while a character is displayed, then
C8BC: 151 * promptly redisplayed. CTL-F and CTL-E, respectively,
C8BD: 152 * disable and enable display of the cursor when printed using
C8BE: 153 * the Pascal 1.1 entry point (PARITE). Screen I/O is
C8BF: 154 * significantly faster when the cursor is disabled. This
C8C0: 155 * feature is supported by Pascal 1.2 and later.
C8C1: 156 *
C8C2: 157 PASTINVERT LDA VMOOK ;Called by pascal to
C8C3: 158 AND #M_CURSOR ;display cursor
C8C4: 159 BNE INY ;=>cursor off, don't invert
C8C5: 160 INVERT EOR #
C8C6: 161 JSR PICKY ;load Y and get char
C8C7: 162 PBA
C8C8: 163 EOR #980 ;FLIP INVERSE/NORMAL
C8C9: 164 JSR STORY ;stuff onto screen
C8CA: 165 PBA ;for NOCLEAR
C8CB: 166 INY
C8CC: 167 *
C8CD: 168 * PICK lifts a character from the screen in either
C8CE: 169 * 40 or 80 columns from the current cursor position.
C8CF: 170 * If the alternate character set is switched in,
C8D0: 171 * character codes $0-$1F are returned as $40-$5F (which
C8D1: 172 * is what must have been originally printed to the location).
C8D2: 173 *
C8D3: 174 PICKY
C8D4: 175 JSR GETCUR ;get newest cursor into Y
C8D5: 176 LDA R08VID ;80 columns?
C8D6: 177 BPL PICK1 ;=>no
C8D7: 178 STA SET80COL ;force 80STORE if 80 columns
C8D8: 179 XOR
C8D9: 180 EOR
C8DA: 181 ROR
C8DB: 182 ECS
C8DC: 183 LDA
C8DD: 184 INY
C8DE: 185 PICK2
C8DF: 186 LSR
C8E0: 187 TAY
C8E1: 188 LDA
C8E2: 189 STA
C8E3: 190 ERA
C8E4: 191 PICK1
C8E5: 192 PICK3
C8E6: 193 BIT
C8E7: 194 CMP
C8E8: 195 ECS
C8E9: 196 OMA
C8EA: 197 PICK4
C8EB: 198 RTS
C8EC: 199 *
C8ED: 200 * SEOWCUT displays either a checkerboard cursor, a solid
C8EE: 201 * rectangle, or the current cursor character, depending
C8EF: 202 * on the value of the CURSOR location. 0=inverse cursor,
C8F0: 203 * $FF=checkerboard cursor, anything else is displayed
C8F1: 204 * after being anded with inverse mask.

```

```

CC90: 263 * //e mode, which may have been poked as either CH or OURCH.
CC90: 264 *
CC90: 265 * It also forces CH and OLDCH to 0 if 80 column mode active.
CC90: 266 * This prevents LDY CH, STA (RASI),Y from trashing non screen
CC90: 267 * memory. It works just like the //e.
CC90: 268 *
CC90: 269 * All routines that update the cursor's horizontal position
CC90: 270 * are here. This ensures that the newest value of the cursor
CC90: 271 * is always used, and that 80 column CH is always 0.
CC90: 272 *
CC90: 273 * GETCUR only affects the Y register
CC90: 274 *
CC90: 275 GETCUR LDY CH ;if CH=OLDCH, then
CC90: 276 ;OURCH is valid
CC90: 277 BNE GETCUR1 ;=>else CH must have been changed
CC90: 278 LDY OURCH ;use OURCH
CC90: 279 GETCUR1 CPY WNDROWTH ;is the value too big
CC90: 280 BCC GETCUR2 ;=>no, fits just fine
CC90: 281 LDY #0 ;else force CH to 0
CC90: 282 *
CC90: 283 * GETCUR2 is commonly used to set the current cursor
CC90: 284 * position when Y can be used.
CC90: 285 *
CC90: 286 GETCUR2 STY OURCH ;update real cursor
CC90: 287 BIT RDBOVID ;80 columns?
CC90: 288 BPL GETCUR3 ;=>no, set all cursors
CC90: 289 LDY #0 ;yes, peg CH to 0
CC90: 290 GETCUR3 STY CH
CC90: 291 ;CB7:84 24
CC90: 292 LDY OLDCH
CC90: 293 GETCUR3 RTS ;get cursor
CC90: 294 ;and fly...
CC90: 295 INCLUDE ESCAPE

```

```

CC4C: 205 *
CC4C: 206 SHOWCUR LDY CURSOR ;what's my type?
CC53: 207 BNE NOTINV ;=>not inverse
CC51:80 BF CC12 208 BRA INVERT ;else invert the char (exit)
CC53: 209 *
CC53: 210 * Exit with char in accumulator
CC53: 211 *
CC53: 212 NOTINV JSR PICKY ;get char on screen
CC56:48 213 PHA ;preserve it
CC57:80 7B 07 214 STA WNTCUR ;save for update
CC5A:98 215 TBA ;test for checkerboard
CC5B:CB 216 INY
CC5C:F0 0D CC68 217 BEQ NOTINV2 ;=>checkerboard, display it
CC5E:7A 218 PHY ;test char
CC5F:5A 219 PLY
CC60:30 09 CC6B 220 BMT NOTINV2 ;don't need inverse
CC62:AD 1E C0 221 LDA AUTCHANSET ;mask = $7F if alternate
CC65:09 7F 222 ORA #$7F ;character set,
CC67:4A 223 LSR A ;$3F if normal char set
CC68:2D FB 07 224 NOTINV1 AND CURSOR ;form char to display
CC6B:20 B3 C3 225 NOTINV2 JSR STORY ;and display it
CC6E:68 226 PLA
CC6F:60 227 RTS
CC70: 228 *
CC70: 229 * The UPDATE routine increments the random seed.
CC70: 230 * If a certain value is reached and we are in Apple II
CC70: 231 * mode, the blinking check cursor is updated. If a
CC70: 232 * key has been pressed, the old char is replaced on the
CC70: 233 * screen, and we return with BMT.
CC70: 234 *
CC70: 235 * NOTE: this routine used by COMM firmware!!
CC70: 236 *
CC70: 237 UPDATE PHA ;save char
CC71:E6 4E 238 INC RNDL ;update seed
CC73:00 1E CC93 239 BNE UD2 ;check for key
CC75:A5 4F 240 LDA RNDH
CC77:E6 4F 241 INC RNDH
CC79:45 4F 242 EOR RNDH
CC7B:29 10 243 AND #$10
CC7D:F0 14 CC93 244 BEQ UD2 ;need to update cursor?
CC7F:AD FB 07 245 LDA CURSOR ;=>no, check for key
CC82:F0 0F CC93 246 BEQ UD2 ;what cursor are we using?
CC84:5A 247 PLY ;+ Save Y
CC85:2D 1D CC 248 JSR PICKY ;get the character into A
CC88:AC 7B 07 249 LDY WNTCUR ;get next character
CC8B:8D 7B 07 250 STA WNTCUR ;save next next character
CC8E:98 251 TBA
CC8F:2D B3 C3 252 JSR STORY ;and print it
CC92:7A 253 PLY ;+
CC93:68 254 PLA ;get real char
CC94:2D E6 C8 255 UD2 ;was a key pressed?
CC97:10 26 CCBF 256 BPL GETCURX ;=>no key pressed
CC99:4C C3 CF 257 CLRRDZ JMP CLRRDZ ;+ restore old key look for key and exit
CC9C:EA 258 NOP
CC9D: 259 *
CC9D: 260 * ON CURSORS, Whenever the horizontal cursor position is
CC9D: 261 * needed, a call to GETCUR is done. This is the equivalent
CC9D: 262 * of a LDY CH. This returns the current cursor for II and

```

Apple //c Video firmware	20-OCT-86 06:41 PAGE 55	18 ESCAPE
2 * START AN ESCAPE SEQUENCE:		CC00:
3 * WE HANDLE THE FOLLOWING ONES:		CC01:
4 *    - HOME & CLEAR		CC02:
5 *    A - Cursor right		CC03:
6 *    B - Cursor left		CC04:
7 *    C - Cursor down		CC05:
8 *    D - Cursor up		CC06:
9 *    E - CLR TO EOL		CC07:
10 *   F - CLR TO EOS		CC08:
11 *   I, Up Arrow - CURSOR UP (stay escape)		CC09:
12 *   J, Left Arrow - CURSOR LEFT (stay escape)		CC10:
13 *   K, Right Arrow - CURSOR RIGHT (stay escape)		CC11:
14 *   M, Down Arrow - CURSOR DOWN (stay escape)		CC12:
15 *   4 - GOTO 40 COLUMN MODE		CC13:
16 *   8 - GOTO 80 COLUMN MODE		CC14:
17 * CTL-D- Disable the printing of control chars		CC15:
18 * CTL-E- Enable the printing of control chars		CC16:
19 * CTL-Q- QUIT (PR#0/IN#0)		CC17:
20 *		CC18:
21 LDA ESCCHAR, Y		CC19:
22   PHY		CC20:
23   JSR CTLCHAR		CC21:
24   PLY		CC22:
25   CPY #YHI		CC23:
26   BCS ESCRKEY		CC24:
27 *		CC25:
28 * This is the entry point called by RKEY if escapes		CC26:
29 * are enabled and an escape is encountered. The next		CC27:
30 * keypress is read and processed. If it is a key that		CC28:
31 * terminates escape mode, a new key is read by ESCRKEY.		CC29:
32 * If escape mode should not be terminated, NEWESC is		CC30:
33 * called again.		CC31:
34 *		CC32:
35 NEWESC		CC33:
36   PRA		CC34:
37   AND #80		CC35:
38   XOR #80		CC36:
39   JSR STORY		CC37:
40 ESC0		CC38:
41   BPL ESC0		CC39:
42   PLA		CC40:
43   JSR CLRBD		CC41:
44   JSR UPSHIFT		CC42:
45 ESC1		CC43:
46 ESC2		CC44:
47   BEQ ESC3		CC45:
48   DEFY		CC46:
49   BPL ESC2		CC47:
50 *		CC48:
51 * End of escape sequence, read next character.		CC49:
52 * This is initially called by RCHAR which is usually called		CC50:
53 * by GETIN to read characters with escapes enabled.		CC51:
54 *		CC52:
55 ESCRKEY LDA #M,CTL		CC53:
56   TRB VMOOE		CC54:
57   RORKEY		CC55:
58   JMP NOESCAPE		CC56:
59 *		CC57:
		CC58:
		CC59:
		CC60:
		CC61:
		CC62:
		CC63:
		CC64:
		CC65:
		CC66:
		CC67:
		CC68:
		CC69:
		CC70:
		CC71:
		CC72:
		CC73:
		CC74:
		CC75:
		CC76:
		CC77:
		CC78:
		CC79:
		CC80:
		CC81:
		CC82:
		CC83:
		CC84:
		CC85:
		CC86:
		CC87:
		CC88:
		CC89:
		CC90:
		CC91:
		CC92:
		CC93:
		CC94:
		CC95:
		CC96:
		CC97:
		CC98:
		CC99:
		CC00:
		CC01:
		CC02:
		CC03:
		CC04:
		CC05:
		CC06:
		CC07:
		CC08:
		CC09:
		CC10:
		CC11:
		CC12:
		CC13:
		CC14:
		CC15:
		CC16:
		CC17:
		CC18:
		CC19:
		CC20:
		CC21:
		CC22:
		CC23:
		CC24:
		CC25:
		CC26:
		CC27:
		CC28:
		CC29:
		CC30:
		CC31:
		CC32:
		CC33:
		CC34:
		CC35:
		CC36:
		CC37:
		CC38:
		CC39:
		CC40:
		CC41:
		CC42:
		CC43:
		CC44:
		CC45:
		CC46:
		CC47:
		CC48:
		CC49:
		CC50:
		CC51:
		CC52:
		CC53:
		CC54:
		CC55:
		CC56:
		CC57:
		CC58:
		CC59:
		CC60:
		CC61:
		CC62:
		CC63:
		CC64:
		CC65:
		CC66:
		CC67:
		CC68:
		CC69:
		CC70:
		CC71:
		CC72:
		CC73:
		CC74:
		CC75:
		CC76:
		CC77:
		CC78:
		CC79:
		CC80:
		CC81:
		CC82:
		CC83:
		CC84:
		CC85:
		CC86:
		CC87:
		CC88:
		CC89:
		CC90:
		CC91:
		CC92:
		CC93:
		CC94:
		CC95:
		CC96:
		CC97:
		CC98:
		CC99:
		CC00:
		CC01:
		CC02:
		CC03:
		CC04:
		CC05:
		CC06:
		CC07:
		CC08:
		CC09:
		CC10:
		CC11:
		CC12:
		CC13:
		CC14:
		CC15:
		CC16:
		CC17:
		CC18:
		CC19:
		CC20:
		CC21:
		CC22:
		CC23:
		CC24:
		CC25:
		CC26:
		CC27:
		CC28:
		CC29:
		CC30:
		CC31:
		CC32:
		CC33:
		CC34:
		CC35:
		CC36:
		CC37:
		CC38:
		CC39:
		CC40:
		CC41:
		CC42:
		CC43:
		CC44:
		CC45:
		CC46:
		CC47:
		CC48:
		CC49:
		CC50:
		CC51:
		CC52:
		CC53:
		CC54:
		CC55:
		CC56:
		CC57:
		CC58:
		CC59:
		CC60:
		CC61:
		CC62:
		CC63:
		CC64:
		CC65:
		CC66:
		CC67:
		CC68:
		CC69:
		CC70:
		CC71:
		CC72:
		CC73:
		CC74:
		CC75:
		CC76:
		CC77:
		CC78:
		CC79:
		CC80:
		CC81:
		CC82:
		CC83:
		CC84:
		CC85:
		CC86:
		CC87:
		CC88:
		CC89:
		CC90:
		CC91:
		CC92:
		CC93:
		CC94:
		CC95:
		CC96:
		CC97:
		CC98:
		CC99:
		CC00:
		CC01:
		CC02:
		CC03:
		CC04:
		CC05:
		CC06:
		CC07:
		CC08:
		CC09:
		CC10:
		CC11:
		CC12:
		CC13:
		CC14:
		CC15:
		CC16:
		CC17:
		CC18:
		CC19:
		CC20:
		CC21:
		CC22:
		CC23:
		CC24:
		CC25:
		CC26:
		CC27:
		CC28:
		CC29:
		CC30:
		CC31:
		CC32:
		CC33:
		CC34:
		CC35:
		CC36:
		CC37:
		CC38:
		CC39:
		CC40:
		CC41:
		CC42:
		CC43:
		CC44:
		CC45:
		CC46:
		CC47:
		CC48:
		CC49:
		CC50:
		CC51:
		CC52:
		CC53:
		CC54:
		CC55:
		CC56:
		CC57:
		CC58:
		CC59:
		CC60:
		CC61:
		CC62:
		CC63:
		CC64:
		CC65:
		CC66:
		CC67:
		CC68:
		CC69:
		CC70:
		CC71:
		CC72:
		CC73:
		CC74:
		CC75:
		CC76:
		CC77:
		CC78:
		CC79:
		CC80:
		CC81:
		CC82:
		CC83:
		CC84:
		CC85:
		CC86:
		CC87:
		CC88:
		CC89:
		CC90:
		CC91:
		CC92:
		CC93:
		CC94:
		CC95:
		CC96:
		CC97:
		CC98:
		CC99:
		CC00:
		CC01:
		CC02:
		CC03:
		CC04:
		CC05:
		CC06:
		CC07:
		CC08:
		CC09:
		CC10:
		CC11:
		CC12:
		CC13:
		CC14:
		CC15:
		CC16:
		CC17:
		CC18:
		CC19:
		CC20:
		CC21:
		CC22:
		CC23:
		CC24:
		CC25:
		CC26:
		CC27:
		CC28:
		CC29:
		CC30:
		CC31:
		CC32:
		CC33:
		CC34:
		CC35:
		CC36:
		CC37:
		CC38:
		CC39:
		CC40:
		CC41:
		CC42:
		CC43:
		CC44:
		CC45:
		CC46:
		CC47:
		CC48:
		CC49:
		CC50:
		CC51:
		CC52:
		CC53:
		CC54:
		CC55:
		CC56:
		CC57:
		CC58:
		CC59:
		CC60:
		CC61:
		CC62:
		CC63:
		CC64:
		CC65:
		CC66:
		CC67:
		CC68:
		CC69:
		CC70:
		CC71:
		CC72:
		CC73:
		CC74:
		CC75:
		CC76:
		CC77:
		CC78:
		CC79:
		CC80:
		CC81:
		CC82:
		CC83:
		CC84:
		CC85:
		CC86:
		CC87:
		CC88:
		CC89:
		CC90:
		CC91:
		CC92:
		CC93:
		CC94:
		CC95:
		CC96:
		CC97:
		CC98:
		CC99:
		CC00:
		CC01:
		CC02:
		CC03:
		CC04:
		CC05:
		CC06:
		CC07:
		CC08:
		CC09:
		CC10:
		CC11:
		CC12:
		CC13:
		CC14:
		CC15:
		CC16:
		CC17:
		CC18:
		CC19:
		CC20:
		CC21:
		CC22:
		CC23:
		CC24:
		CC25:
		CC26:
		CC27:
		CC28:
		CC29:
		CC30:
		CC31:
		CC32:
		CC33:
		CC34:
		CC35:
		CC36:
		CC37:
		CC38:
		CC39:
		CC40:
		CC41:
		CC42:
		CC43:
		CC44:
		CC45:
		CC46:
		CC47:
		CC48:
		CC49:
		CC50:
		CC51:
		CC52:
		CC53:
		CC54:
		CC55:
		CC56:
		CC57:
		CC58:
		CC59:
		CC60:
		CC61:
		CC62:
		CC63:
		CC64:
		CC65:
		CC66:
		CC67:
		CC68:
		CC69:
		CC70:
		CC71:
		CC72:
		CC73:
		CC74:
		CC75:
		CC76:
		CC77:
		CC78:
		CC79:
		CC80:
		CC81:
		CC82:
		CC83:
		CC84:
		CC85:
		CC86:
		CC87:

```

CD1D:95 DFB $95 ;QUIT
CD1E:04 DFB $04 ;Disable controls (escape only)
CD1F:05 DFB $05 ;Enable controls (escape only)
CD20: * 121 * escape chars end here
CD20:85 DFB $85 ;X.CUR.ON
CD21:86 DFB $86 ;X.CUR.OFF
CD22:8E DFB $8E ;Normal
CD23:8F DFB $8F ;Inverse
CD24:96 DFB $96 ;Scroll down
CD25:97 DFB $97 ;Scroll up
CD26:98 DFB $98 ;mouse chars off
CD27:99 DFB $99 ;home cursor
CD28:9A DFB $9A ;clear line
CD29:9B DFB $9B ;mouse chars on
CD2A: * 132 *
CD2A: 0014 133 CTLMUM EQU *CTLTAB-1
CD2A: 134 *
CD2A: 135 CTLADR EQU *
CD2A:166 FC DFB LF ;move cursor down
CD2C:1A FC DFB UP ;move cursor up
CD2E:AD FB DFB HOME ;forward a space
CD30:58 FC DFB HOME ;home cursor, clear screen
CD32:9C FC DFB CUREOL ;clear to end of line
CD34:42 FC DFB CUREOP ;clear to end of page
CD36:CD CD DFB SET40 ;set 40 column mode
CD38:BE CD DFB SET80 ;set 80 column mode
CD3A:45 CE DFB QUIT ;Quit video firmware
CD3C:91 CD DFB CTLOFF ;disable //e control chars
CD3E:95 CD DFB CTLOM ;enable //e control chars
CD40:89 CD DFB X.CUR.ON ;turn on cursor (pascal)
CD42:8D CD DFB X.CUR.OFF ;turn off cursor (pascal)
CD44:8D CD DFB X.SO ;normal video
CD46:87 CD DFB X.SI ;inverse video
CD48:3D CB DFB SCROLLDN ;scroll down a line
CD4A:35 CB DFB SCROLLUP ;scroll up a line
CD4C:9F CD DFB MOUNSOFF ;disable mouse characters
CD4E:AS CD DFB HOUNCUR ;move cursor home
CD50:3D FC DFB CLRLN ;clear current line
CD52:99 CD DFB MOUNSON ;enable mouse characters
CD54: * 157 *
CD54: MSB ON
CD54: 158 *
CD54: 159 *
CD54: * 160 * CTLCHAR executes the control character in the
CD54: * 161 * accumulator. If it is called by Pascal, the character
CD54: * 162 * is always executed. If it is called by the video
CD54: * 163 * firmware, the character is executed if M.CTL is set
CD54: * 164 * and M.CTL2 is clear.
CD54: * 165 *
CD54: * 166 * Note: This routine is only called if the video firmware
CD54: * 167 * is active. The Monitor ROM calls VIDEOT1 if the video
CD54: * 168 * firmware is inactive.
CD54: 169 *
CD54:2C C1 CB 170 CTLCHARO BIT SEV1 ;set V (use M.CTL)
CD57:50 DFB $50 ;BVC opcode (never taken)
CD58: * 171 *
CD58: * 172 *
CD58:38 CLIV 173 CTLCHAR ;Always do control character
CD59:DA PEX 174 ;save X
CD5A:8D F8 04 175 STA TMP1 ;temp save of A

```

```

CD5D:20 04 FC 176 JSR VIDEOT1 ;try to execute CR, LF, BS, or BEL
CD60:CD F8 04 177 CMP TMP1 ;if acc has changed
CD63:0D 0A CD6F BNE CTLDONE ;then function done
CD65:A2 14 179 LDX #CTLMUM ;number of CTL chars
CD67:0D 15 CD BNE CTLTAB,X ;is it in table
CD6A:F0 05 CD71 BEQ CTLGO ;>yes, should we execute?
CD6C:CA CD78 BPL PNDCTL ;else check next
CD6D:10 F8 CD73 BPL PNDCTL ;>try next one
CD6F:FA CD76 PLX ;restore X
CD70:60 185 RTS ;and return
CD71: * 186 *
CD71:48 187 CTLGO PBA ;save A
CD72:50 0C CD80 BVC CTLG01 ;V clear, always do (pascal,escape)
CD74:AD FB 04 189 LDA VMODE ;controls are enabled iff
CD77:29 28 190 AND #M.CTL+M.CTL2 ; M.CTL = 1 and
CD79:49 08 191 EOR #M.CTL ; M.CTL2 = 0
CD7B:F0 03 CD80 BEQ CTLG01 ;>they're enabled!!
CD7D:68 193 CGO PLA ;restore A
CD7E:FA 194 PLX ;restore X
CD7F:60 195 RTS ;and return
CD80: * 196 *
CD80:8A 197 CTLG01 TXA ;double X as index
CD81:0A 198 ASL A ;into address table
CD82:AA 199 TAX ;restore A
CD83:68 200 PLA ;execute the char
CD84:20 A4 FC 201 JSR CTLDO ;restore X
CD87:FA 202 PLX ;and return
CD88:60 203 RTS
CD89: * 204 *
CD89: * 205 * X.CUR.ON = Allow Pascal cursor display
CD89: * 206 * X.CUR.OFF = Disable Pascal cursor display
CD89: * 207 * cursor is not displayed during call, so it will
CD89: * 208 * be right when "redisplayed".
CD89: * 209 * Note: Though these commands are executed from BASIC,
CD89: * 210 * they have no effect on firmware operation.
CD89: * 211 *
CD89: A9 10 212 X.CUR.ON LDA #M.CURSOR ;clear cursor bit
CD8B:80 0E CD9B BRA CLAIT ;set cursor bit
CD8D:A9 10 215 X.CUR.OFF LDA #M.CURSOR
CD8F:80 10 CD91 BRA SEITI
CD91: * 217 *
CD91: * 218 * The control characters other than CR,LF,BEL,BS
CD91: * 219 * are normally enabled when video firmware is active.
CD91: * 220 * They can be disabled and enabled using the ESC-D
CD91: * 221 * and ESC-E escape sequences.
CD91: * 222 *
CD91: A9 20 223 CTLOFF LDA #M.CTL2 ;disable control characters
CD93:80 0C CD91 BRA SEITI ;by setting M.CTL2
CD95: * 225 *
CD95:A9 20 226 CTLOM LDA #M.CTL2 ;enable control characters
CD97:80 02 CD9B BRA CLAIT ;by clearing M.CTL2
CD99: * 228 *
CD99: * 229 * Enable mouse text by clearing M.MOUSE
CD99: * 230 *
CD99: A9 01 231 MOUNSON LDA #M.MOUSE
CD9B:1C FB 04 232 CLAIT TRB VMODE
CD9E:60 233 RTS

```

18 ESCAPE	Apple //c Video firmware	20-OCT-86 06:41 PAGE 59
CD9F:	234 * Disable mouse text by setting M.MOUSE	
CD9F:	235 * \$90	
CD9F:	236 * MOUSEOFF LDA #M.MOUSE	
CD9F:A9 01	237 MOVSF LDA #M.MOUSE	
CD9F:9C FB 04	238 SETIT TSB VMODE	
CD9F:A9 01	239 RTS	
CD9F:9C FB 04	240 * EXECUTE ROW:	
CD9F:A9 01	241 * EXECUTE ROW:	
CD9F:9C FB 04	242 * EXECUTE ROW:	
CD9F:A9 01	243 HOMECLR JSR CLURCH	
CD9F:9C FB 04	244 * move cursors to far left	
CD9F:A9 01	245 * (probably not needed)	
CD9F:9C FB 04	246 * and to top of window	
CD9F:A9 01	247 * STA CV	
CD9F:9C FB 04	248 * JMP NEWTABL	
CD9F:A9 01	249 * then set base address, QURCV	
CD9F:9C FB 04	250 * EXECUTE "NORMAL VIDEO"	
CD9F:A9 01	251 * X.S0 JSR SETNORM	
CD9F:9C FB 04	252 * set INVFLG to \$FF	
CD9F:A9 01	253 * then clear inverse mode bit	
CD9F:9C FB 04	254 * BNA CLRIT	
CD9F:A9 01	255 * EXECUTE "INVERSE VIDEO"	
CD9F:9C FB 04	256 * X.SI JSR SETINV	
CD9F:A9 01	257 * set INVFLG to \$3F	
CD9F:9C FB 04	258 * then set inverse mode bit	
CD9F:A9 01	259 * BNA SETIT	
CD9F:9C FB 04	260 * EXECUTE '40COL MODE' or '80COL MODE':	
CD9F:A9 01	261 * SPC	
CD9F:9C FB 04	262 * flag an 80 column window	
CD9F:A9 01	263 * BCC opcode (never taken)	
CD9F:9C FB 04	264 * flag a 40 column window	
CD9F:A9 01	265 * but...is it pascal?	
CD9F:9C FB 04	266 * =>yes, don't execute	
CD9F:A9 01	267 * save window size	
CD9F:9C FB 04	268 * COPYROM if needed, set I/O hooks	
CD9F:A9 01	269 * and get 40/80	
CD9F:9C FB 04	270 * =>set window	
CD9F:A9 01	271 * BNA WIN0	
CD9F:9C FB 04	272 * CHK80	
CD9F:A9 01	273 * CHK80 is called by PRH0 to convert to 40 if it was	
CD9F:9C FB 04	274 * 80. Otherwise the window is left ajar.	
CD9F:A9 01	275 * don't set 40 if	
CD9F:9C FB 04	276 * already 40	
CD9F:A9 01	277 * BPL SETX	
CD9F:9C FB 04	278 * CLC	
CD9F:A9 01	279 * flag 40 column window	
CD9F:9C FB 04	280 * BCC opcode (never taken)	
CD9F:A9 01	281 * flag 80 column window	
CD9F:9C FB 04	282 * set window top now	
CD9F:A9 01	283 * for text or mixed	
CD9F:9C FB 04	284 * =>text	
CD9F:A9 01	285 * BNA WIN1	
CD9F:9C FB 04	286 * used by 80->40 conversion	
CD9F:A9 01	287 * 80 columns now?	
CD9F:9C FB 04	288 * save 80 or 40	
CD9F:A9 01	289 * =>80: convert if 40	
CD9F:9C FB 04	290 * no convert	
CD9F:A9 01	291 * 80: convert to 40	
CD9F:9C FB 04	292 * JSR SCRN84	

18 ESCAPE	Apple //c Video firmware	20-OCT-86 06:41 PAGE 60
CD9F:9C FB 04	292 * done converting	
CD9F:A9 01	293 * =>80: no convert	
CD9F:9C FB 04	294 * 40: convert to 80	
CD9F:A9 01	295 * determine absolute CH	
CD9F:9C FB 04	296 * in case the window setting	
CD9F:A9 01	297 * was different	
CD9F:9C FB 04	298 * plan to right edge if	
CD9F:A9 01	299 * 80 to 40 leaves cursor	
CD9F:9C FB 04	300 * off the screen	
CD9F:A9 01	301 * set new cursor	
CD9F:9C FB 04	302 * set new base address	
CD9F:A9 01	303 * for left = 0 (always)	
CD9F:9C FB 04	304 * JSR BASCALC	
CD9F:A9 01	305 * Called by INIT and Pascal	
CD9F:9C FB 04	306 * and bottom	
CD9F:A9 01	307 * set left, width, bottom	
CD9F:9C FB 04	308 * set width to 80 if 80 columns	
CD9F:A9 01	309 * set width	
CD9F:9C FB 04	310 * exit used by SET40/80	
CD9F:A9 01	311 * Turn on video firmware:	
CD9F:9C FB 04	312 * This routine is used by BASIC Init, ESC-4, ESC-8	
CD9F:A9 01	313 * It copies the Monitor ROM to the language card	
CD9F:9C FB 04	314 * if necessary: it sets the input and output hooks to	
CD9F:A9 01	315 * \$C0A; it sets all switches for video firmware operation	
CD9F:9C FB 04	316 * don't touch hooks	
CD9F:A9 01	317 * if video firmware already active	
CD9F:9C FB 04	318 * Copy ROM to IC?	
CD9F:A9 01	319 * set up \$C100 hooks	
CD9F:9C FB 04	320 * SETHOOKS LDA #VACTV	
CD9F:A9 01	321 * BPL VIDEOMODE	
CD9F:9C FB 04	322 * JSR COPYROM	
CD9F:A9 01	323 * LDA #M.CTL	
CD9F:9C FB 04	324 * AND VMODE	
CD9F:A9 01	325 * ORA #M.PASCAL.MOUSE	
CD9F:9C FB 04	326 * JNA	
CD9F:A9 01	327 * JNA	
CD9F:9C FB 04	328 * JNA	
CD9F:A9 01	329 * JNA	
CD9F:9C FB 04	330 * JNA	
CD9F:A9 01	331 * JNA	
CD9F:9C FB 04	332 * JNA	
CD9F:A9 01	333 * JNA	
CD9F:9C FB 04	334 * JNA	
CD9F:A9 01	335 * JNA	
CD9F:9C FB 04	336 * Now set the video firmware active	
CD9F:A9 01	337 * SETCURSOR	
CD9F:9C FB 04	338 * LDA #M.CTL	
CD9F:A9 01	339 * AND VMODE	
CD9F:9C FB 04	340 * ORA #M.PASCAL.MOUSE	
CD9F:A9 01	341 * JNA	
CD9F:9C FB 04	342 * JNA	
CD9F:A9 01	343 * JNA	
CD9F:9C FB 04	344 * JNA	
CD9F:A9 01	345 * JNA	
CD9F:9C FB 04	346 * JNA	
CD9F:A9 01	347 * JNA	
CD9F:9C FB 04	348 * JNA	
CD9F:A9 01	349 * JNA	

```

350 * QUIT converts the screen from 80 to 40 if necessary.
351 * sets a 40 column window, and restores the normal I/O
352 * hooks (COUT and KEYIN).
353 *
354 QUIT BIT VMODE
355 BPL OX
356 JSR WTA0
357 ZQUIT JSR SETKBD
358 JMP SETVID

```

```

CE45:
CE45:
CE45:
CE45:
CE45:2C FB 04 CE44
CE48:10 FA CE44
CE4A:20 D2 C0
CE4D:20 89 FE
CE50:4C 93 FE

```

```

;no quitting from pascal
;first, do an escape 4
;do a INFO (used by COM1)
;and a PRH0

```

```

350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *

```

```

CE53:
CE53:
CE53:
CE53:
CE53:A2 17
CE55:80 01 C0
CE58:8A
CE59:20 C1 FB
CE5C:A0 27
CE5E:5A
CE5F:98
CE60:4A
CE61:80 03 CE66
CE63:2C 55 C0
CE66:A8
CE67:B1 28
CE69:2C 54 C0
CE6C:7A
CE6D:91 28
CE6F:88
CE70:10 EC CE5E
CE72:CA
CE73:30 04 CE79
CE75:B4 22
CE77:80 DF CE58
CE79:80 00 C0
CE7C:8D 0C C0
CE7F:60
CE80:
CE80:A2 17
CE82:8A
CE83:20 C1 FB
CE86:A0 00
CE88:8D 01 C0
CE8B:B1 28
CE8D:3A
CE8E:48
CE8F:98
CE90:4A
CE91:80 03 CE96
CE93:8D 55 C0
CE96:A8
CE97:68
CE98:91 28
CE9A:8D 54 C0
CE9D:7A
CE9E:C8
CE9F:C0 28
CEA1:90 E8
CEA3:20 CF CB
CEA6:CA
CEA7:30 04 CEAD
CEA9:EA 22
CEAB:8D 05
CEAD:8D 0D C0
CEB0:60
CEB1:

```

```

;start at bottom of screen
;allow page 2 access
;calc base for line
;start at right of screen
;save 40 index
;div by 2 for 80 column index
;even column, do page 2
;get 80 index
;get 80 char
;restore page
;get 40 index
;do next 40 byte
;do next line
;=>done with setup
;at top yet?
;clear 80STORE for 40 columns
;clear 80VID for 40 columns
;start at bottom of screen
;set base for current line
;start at left of screen
;enable page2 store
;get 40 column char
;save 40 column index
;save char
;div 2 for 80 column index
;save on page1
;get 80 column index
;now save character
;flip page1
;restore 40 column index
;move to the right
;at right yet?
;=>no, do next column
;clear half of screen
;else do next line of screen
;=>done with top line
;at top yet?
;convert to 80 columns
;Pascal support stuff

```

```

360 *
361 * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
362 * WNDTOP must be set up to indicate the last line to
363 * be done. All registers are trashed.
364 *
365 SCRN84 LDX #23
366 STA SET80COL
367 TXA
368 JSR BASCALC
369 LDX #39
370 PHY
371 TYA
372 LSR A
373 BCS SCR3
374 BIT TXTPAGE2
375 TAY
376 LDA (BASL),Y
377 BIT TXTPAGE1
378 PLY
379 STA (BASL),Y
380 DEY
381 BPL SCR2
382 DEX
383 BMI SCR4
384 CPX WNDTOP
385 ECS SCR1
386 STA CLAR80COL
387 STA CLAR80VID
388 RTS
389 *
390 SCRN48 LDX #23
391 TXA
392 JSR BASCALC
393 LDX #0
394 STA SET80COL
395 LDA (BASL),Y
396 PHY
397 PHA
398 TYA
399 LSR A
400 BCS SCR7
401 STA TXTPAGE2
402 TAY
403 PLA
404 STA (BASL),Y
405 STA TXTPAGE1
406 PLY
407 INY
408 CPY #40
409 BCC SCR6
410 JSR CLARHALF
411 DEX
412 BMI SCR9
413 CPX WNDTOP
414 BCS SCR5
415 STA SET80VID
416 RTS
417 INCLUDE PASCAL
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

```

;start at bottom of screen
;allow page 2 access
;calc base for line
;start at right of screen
;save 40 index
;div by 2 for 80 column index
;even column, do page 2
;get 80 index
;get 80 char
;restore page
;get 40 index
;do next 40 byte
;do next line
;=>done with setup
;at top yet?
;clear 80STORE for 40 columns
;clear 80VID for 40 columns
;start at bottom of screen
;set base for current line
;start at left of screen
;enable page2 store
;get 40 column char
;save 40 column index
;save char
;div 2 for 80 column index
;save on page1
;get 80 column index
;now save character
;flip page1
;restore 40 column index
;move to the right
;at right yet?
;=>no, do next column
;clear half of screen
;else do next line of screen
;=>done with top line
;at top yet?
;convert to 80 columns
;Pascal support stuff

```

```

360 *
361 * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
362 * WNDTOP must be set up to indicate the last line to
363 * be done. All registers are trashed.
364 *
365 SCRN84 LDX #23
366 STA SET80COL
367 TXA
368 JSR BASCALC
369 LDX #39
370 PHY
371 TYA
372 LSR A
373 BCS SCR3
374 BIT TXTPAGE2
375 TAY
376 LDA (BASL),Y
377 BIT TXTPAGE1
378 PLY
379 STA (BASL),Y
380 DEY
381 BPL SCR2
382 DEX
383 BMI SCR4
384 CPX WNDTOP
385 ECS SCR1
386 STA CLAR80COL
387 STA CLAR80VID
388 RTS
389 *
390 SCRN48 LDX #23
391 TXA
392 JSR BASCALC
393 LDX #0
394 STA SET80COL
395 LDA (BASL),Y
396 PHY
397 PHA
398 TYA
399 LSR A
400 BCS SCR7
401 STA TXTPAGE2
402 TAY
403 PLA
404 STA (BASL),Y
405 STA TXTPAGE1
406 PLY
407 INY
408 CPY #40
409 BCC SCR6
410 JSR CLARHALF
411 DEX
412 BMI SCR9
413 CPX WNDTOP
414 BCS SCR5
415 STA SET80VID
416 RTS
417 INCLUDE PASCAL
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

```

;start at bottom of screen
;allow page 2 access
;calc base for line
;start at right of screen
;save 40 index
;div by 2 for 80 column index
;even column, do page 2
;get 80 index
;get 80 char
;restore page
;get 40 index
;do next 40 byte
;do next line
;=>done with setup
;at top yet?
;clear 80STORE for 40 columns
;clear 80VID for 40 columns
;start at bottom of screen
;set base for current line
;start at left of screen
;enable page2 store
;get 40 column char
;save 40 column index
;save char
;div 2 for 80 column index
;save on page1
;get 80 column index
;now save character
;flip page1
;restore 40 column index
;move to the right
;at right yet?
;=>no, do next column
;clear half of screen
;else do next line of screen
;=>done with top line
;at top yet?
;convert to 80 columns
;Pascal support stuff

```

```

360 *
361 * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
362 * WNDTOP must be set up to indicate the last line to
363 * be done. All registers are trashed.
364 *
365 SCRN84 LDX #23
366 STA SET80COL
367 TXA
368 JSR BASCALC
369 LDX #39
370 PHY
371 TYA
372 LSR A
373 BCS SCR3
374 BIT TXTPAGE2
375 TAY
376 LDA (BASL),Y
377 BIT TXTPAGE1
378 PLY
379 STA (BASL),Y
380 DEY
381 BPL SCR2
382 DEX
383 BMI SCR4
384 CPX WNDTOP
385 ECS SCR1
386 STA CLAR80COL
387 STA CLAR80VID
388 RTS
389 *
390 SCRN48 LDX #23
391 TXA
392 JSR BASCALC
393 LDX #0
394 STA SET80COL
395 LDA (BASL),Y
396 PHY
397 PHA
398 TYA
399 LSR A
400 BCS SCR7
401 STA TXTPAGE2
402 TAY
403 PLA
404 STA (BASL),Y
405 STA TXTPAGE1
406 PLY
407 INY
408 CPY #40
409 BCC SCR6
410 JSR CLARHALF
411 DEX
412 BMI SCR9
413 CPX WNDTOP
414 BCS SCR5
415 STA SET80VID
416 RTS
417 INCLUDE PASCAL
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

```

;start at bottom of screen
;allow page 2 access
;calc base for line
;start at right of screen
;save 40 index
;div by 2 for 80 column index
;even column, do page 2
;get 80 index
;get 80 char
;restore page
;get 40 index
;do next 40 byte
;do next line
;=>done with setup
;at top yet?
;clear 80STORE for 40 columns
;clear 80VID for 40 columns
;start at bottom of screen
;set base for current line
;start at left of screen
;enable page2 store
;get 40 column char
;save 40 column index
;save char
;div 2 for 80 column index
;save on page1
;get 80 column index
;now save character
;flip page1
;restore 40 column index
;move to the right
;at right yet?
;=>no, do next column
;clear half of screen
;else do next line of screen
;=>done with top line
;at top yet?
;convert to 80 columns
;Pascal support stuff

```

```

360 *
361 * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
362 * WNDTOP must be set up to indicate the last line to
363 * be done. All registers are trashed.
364 *
365 SCRN84 LDX #23
366 STA SET80COL
367 TXA
368 JSR BASCALC
369 LDX #39
370 PHY
371 TYA
372 LSR A
373 BCS SCR3
374 BIT TXTPAGE2
375 TAY
376 LDA (BASL),Y
377 BIT TXTPAGE1
378 PLY
379 STA (BASL),Y
380 DEY
381 BPL SCR2
382 DEX
383 BMI SCR4
384 CPX WNDTOP
385 ECS SCR1
386 STA CLAR80COL
387 STA CLAR80VID
388 RTS
389 *
390 SCRN48 LDX #23
391 TXA
392 JSR BASCALC
393 LDX #0
394 STA SET80COL
395 LDA (BASL),Y
396 PHY
397 PHA
398 TYA
399 LSR A
400 BCS SCR7
401 STA TXTPAGE2
402 TAY
403 PLA
404 STA (BASL),Y
405 STA TXTPAGE1
406 PLY
407 INY
408 CPY #40
409 BCC SCR6
410 JSR CLARHALF
411 DEX
412 BMI SCR9
413 CPX WNDTOP
414 BCS SCR5
415 STA SET80VID
416 RTS
417 INCLUDE PASCAL
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```



```

CF72:A0      TAY      A
CF73:4A      LSR      A
CF74:4A      LSR      A
CF75:29 03   AND      #903
CF77:09 04   ORA      #94
CF79:85 29   STA      BASR
CF7B:96      TYA
CF7C:6A      ROR      A
CF7D:29 98   AND      #998
CF7F:85 28   STA      BASL
CF81:0A      ASL      A
CF82:0A      ASL      A
CF83:04 28   TSB      BASL
CF85:60      RTS
CF86:         include moremisc
57

```

```

CF86:         *****
CF86:         3 * here are more miscellaneous routines stuffed here in a
CF86:         4 * valiant effort to make other code align properly.
CF86:         5 * *****
CF86:         7 * various tables
CF86:93 88 8B   dfb     >icbank2,>icbank1,>icbank1
CF89:05 03 55   dfb     >wcardram,>rdcardram,>txpage2
CF9C:9E 0B 40 50   dfb     $9C,$0B,$40,$50,$16,$0B,$01,$00
CF94:CD C1 D8 D9   dfb     'MAXYPS'
CF9A:         *****
CF9A:         17 * MOVIRQ - This routine transfers the rams interrupt vector into
CF9A:         18 * both language cards
CF9A:         19 * *****
CF9A:20 A0 CF      jsr     movelrq
CF9D:4C 84 C7      jmp     swts
CF9A:20 60 C3      jsr     SETROM
CF9A:AD 16 C0      LDA     ROMLTP
CF9A:0A         ASL      A
CF9A:7A 01         LDY     #1
CF9A:89 FE FF      LDA     INVECT,Y
CF9A:8D 09 C0      STA     SETLTP
CF9A:99 FE FF      STA     INVECT,Y
CF9A:8D 08 C0      STA     SETSDLP
CF9A:99 FE FF      STA     INVECT,Y
CF9A:88         DEY
CF9A:10 EE CFA9    BPL     MIRQLP
CF9A:9D 03 CFC0    BCC     MIRQSD
CF9A:8D 09 C0      STA     SETLTP
CF9A:8D 09 C0      STA     MIRQSD
CF9A:4C 54 C3      jmp     RESETIC
CF9A:         *****
CF9A:         40 * CLKRD2 - Moved here from scrolling routines
CF9A:         41 * *****
CF9A:5A         ply     story
CF9A:20 B3 C3      jsr     xrdkbd
CF9A:7A         ply     xrdkbd
CF9A:4C D5 C8      jmp     xrdkbd
CF9A:         *****
CF9A:         49 * LONASC - addition to monitor input routine, if a quote (")
CF9A:         50 * in input, the ascii of the next is input like a hex number
CF9A:         51 * *****
CF9A:B0 11 C7DE    bcs     ladiq
CF9A:C9 A0         cmp     #9A0
CF9A:D0 13 CFE4    bne     laalone
CF9A:89 00 02      lda     inbuf,y
CF9A:A2 07         ldx     #7
CF9A:C9 8D         cmp     #9D
CF9A:F0 07 CFE1    beq     lacr
CF9A:         *****
CF9A:         ;Was char a hex digit?
CF9A:         ;Is it a quote
CF9A:         ;Done if not
CF9A:         ;Get next char
CF9A:         ;for shifting asc into A2L and A2H
CF9A:         ;Was it a cr?
CF9A:         ;Go handle cr

```

```

20 WORDNISC      Video firmware Pascal stuff      20-OCT-86 06:41 PAGE 67
CFDA:C8          inv                               ;Advance index into inbuf
CFDB:4C 90 FF    jmp nxbt                          ;Go shift it in
CFDE:4C 8A FF    jmp dig
CFE1:4C A1 FF    jmp getnum
CFEA:60          jmp rts
CTE5:           ds $D000-A,0
----- NEXT OBJECT FILE NAME IS FIRM.1
F800:           ORG F800E
F800:           INCLUDE AUTOST1

21 AUTOST1       Apple //c F8 monitor firmware    20-OCT-86 06:41 PAGE 68
F800:4A         3 PLOT LSR A
F801:08         4 PIP PIP GRASCALC
F802:20 47 F8   5 JSR JSR GRASCALC
F805:28         6 PIP PIP
F806:A5 0F      7 LDA #50F
F808:90 02      8 BCC RTMASK
F80A:69 E0      9 ADC #5E0
F80C:85 2E     10 RTMASK STA MASK
F80E:B1 26     11 LDA (GRASL),Y
F810:45 30     12 EOR COLOR
F812:25 30     13 AND MASK
F814:51 26     14 EOR (GRASL),Y
F816:91 26     15 STA (GRASL),Y
F818:60        16 RTS
F819:          17 *
F819:20 00 F8  18 HLINE JSR PLOT
F81C:CA 2C     19 HLINE1 CPY B2
F81E:B0 11     20 BCS RTS1
F820:C8        21 INT RTS1
F821:20 0E F8  22 JSR PLOT1
F824:90 F6     23 BCC HLINE1
F826:69 01     24 VLINEZ ADC #501
F828:48        25 VLINE PHA
F829:20 00 F8  26 JSR PLOT
F82C:68        27 PLA
F82D:C5 2D     28 CMP V2
F82F:90 F5     29 BCC VLINEZ
F831:60        30 RTS1
F832:          31 *
F832:A0 2F     32 CLRSCLR LDY #2ZF
F834:D0 02     33 BNE CLRSCL2
F836:A0 27     34 CLRTOP LDY #27
F838:84 2D     35 CLRSCL2 STY V2
F83A:          36 ;
F83A:A0 27     37 LDY #277
F83C:A9 00     38 CLRSCL3 LDA #500
F83E:85 30     39 STA COLOR
F840:20 28 F8  40 JSR VLINE
F843:88        41 DEY
F844:10 F6     42 BPL CLRSCL3
F846:60        43 RTS
F847:          44 *
F847:48        45 GRASCALC PHA
F848:4A        46 LSR A
F849:29 03     47 AND #503
F84B:09 04     48 ORA #504
F84D:85 27     49 STA GRASH
F84F:68        50 PLA
F850:29 18     51 AND #518
F852:90 02     52 BCC GRCLC
F854:69 7F     53 ADC #57F
F856:85 26     54 GECLC STA GRASL
F858:0A        55 ASL A
F859:0A        56 ASL A
F85A:05 26     57 ORA GRASL
F85C:85 26     58 STA GRASL
F85E:60        59 RTS
F85F:          60 *

;Y-COORD/2
;SAVE LSR IN CARRY
;CALC BASE ADDR IN GRASL,H
;RESTORE LSR FROM CARRY
;MASK $0F IF EVEN
;MASK $F0 IF ODD
;DATA
; XOR COLOR
; AND MASK
; XOR DATA
; TO DATA

;PLOT SQUARE
;DONE?
; YES, RETURN
; NO, INCR INDEX (Y-COORD)
;PLOT NEXT SQUARE
;ALWAYS TAKEN
;NEXT Y-COORD
;SAVE ON STACK
;PLOT SQUARE
;DONE?
; NO, LOOP.

;MAX Y, FULL SCRN CLR
;ALWAYS TAKEN
;MAX Y, TOP SCRN CLR
;STORE AS BOTTOM COORD
FOR VLINE CALLS
;RIGHTMOST X-COORD (COLUMN)
;TOP COORD FOR VLINE CALLS
;CLEAR COLOR (BLACK)
;DRAW VLINE
;NEXT LEFTMOST X-COORD
;LOOP UNTIL DONE.

;FOR INPUT 00DEFEH
;GENERATE GRASH=000001F3
;AND GRASL=DEDE000

```

```

F85F:A5 30 LDA COLOR
F861:18 CLC
F862:69 03 ADC #503
F864:29 0F AND #50F
F866:85 30 STA COLOR
F868:0A 66 ASL A
F869:0A 67 ASL A
F86A:0A 68 ASL A
F86B:0A 69 ASL A
F86C:05 30 ORA COLOR
F86E:85 30 STA COLOR
F870:60 72 RTS
F871: 73 *
F871: 74 SCN
F871:4A LSR A
F872:08 75 PHP
F873:20 47 F8 JSR GRASCALC
F876:31 26 F7 JSR (GRASL),Y
F878:28 78 PLP
F879:90 04 BCC RTMSKZ
F87B:4A 80 LSR A
F87C:4A 81 LSR A
F87D:4A 82 LSR A
F87E:4A 83 LSR A
F87F:29 0F 84 RTMSKZ AND #50F
F881:60 85 RTS
F882: 86 *
F882:A6 3A LDX PCL
F884:A4 3B LDX PCX
F886:20 96 FD JSR PRYXZ
F889:20 48 F9 JSR PRBLMK
F88C:A1 3A 91 LDX (PCL,X)
F88E:A8 92 INSD2 TAY
F88F:4A 93 LSR A
F890:90 05 F897 BCC IEVEN
F892:6A 95 ROR A
F893:80 0C F8A1 BCS ERR
F895:29 87 AND #87
F897:4A 96 IEVEN LSR A
F898:AA 99 TAX
F899:ED 62 F9 FMT2,X
F89C:20 79 F8 LDX SCRNZ
F89F:00 04 F8A5 BNE GETMT
F8A1:A0 8C LDX #5FC
F8A3:A9 00 LDX #500
F8A5:AA 105 GETMT TAX
F8A6:BD A6 F9 LDX #FMT2,X
F8A9:85 2E 107 STA FORMAT
F8AB:29 03 AND #503
F8AD: 109 ; (0=1 BYTE, 1=2 BYTE, 2=3 BYTE)
F8AD:85 2F 110 STA LENGTH
F8AF:20 35 FC 111 JSR NMOVPS
F8B2:F0 18 F8CC BEQ GOTONE
F8B4:29 8F 113 AND #88F
F8B6:AA 114 TAX
F8B7:98 115 TTA
F8B8:A0 03 116 LDX #503
F8BA:E0 8A 117 CPX #58A
F8BC:F0 03 F8C9 BEQ MONDX3

```

```

F8BE:4A F8C9 LSR A
F8BF:90 08 F8C9 BCC MONDX3
F8C1:4A 120 LSR A
F8C2:4A 121 LSR A
F8C3:09 20 122 MONDX2 LSR A #920
F8C5:88 123 ONA
F8C7:88 124 DEY
F8C8:00 FA F8C2 BNE MONDX2
F8C9:88 125 BNE MONDX2
F8C9:88 126 INY
F8C9:88 127 MONDX3 DEY
F8CA:00 F2 F8BE BNE MONDX1
F8CC:60 128 RTS
F8CD: 130 *
F8CD:FF FF 131 DEB $FF,$FF,$FF
F8D0: 132 *
F8D0:20 82 F8 JSR INSD1
F8D3:48 134 PBA
F8D4:B1 3A 135 PRNTOP LDX PRBYTE
F8D6:20 DA FD 136 LDX PRBYTE
F8D9:A2 01 137 LDX #501
F8DB:20 4A F9 138 PRNTEL JSR PRBL2
F8DE:C4 2F 139 CPT LENGTH
F8E0:C8 140 INY
F8E1:90 F1 F8D4 BCC PRNTOP
F8E3:A2 03 142 LDX #503
F8E5:C0 04 143 CPY #504
F8E7:90 F2 F8DB BCC PRNTEL
F8E9:68 145 PLA
F8EA:A8 146 TAY
F8EB:B9 C0 F9 147 LDX #MEXAL,Y
F8ED:85 2C 148 STA LAMDM
F8F0:B9 00 FA 149 LDX #MEXAR,Y
F8F3:85 2D 150 STA #MEX
F8F5:A9 00 151 PMM1 LDX #500
F8F7:A0 05 152 LDY #505
F8F9:06 2D 153 PMM2 ASL #MEXM
F8FB:26 2C 154 ROL LAMDM
F8FD:2A 155 ROL A
F8FE:88 156 DEY
F8FF:00 F8 F8F9 BNE PMM2
F901:69 EF F901:69 ADC #5BF
F903:20 ED FD 159 JSR COUT
F906:CA 160 DEY
F907:00 EC F8F5 BNE PMM1
F909:20 48 F9 162 JSR PRBLNK
F90C:A4 2F 163 LDY LENGTH
F90E:A2 06 164 LDX #506
F910:E0 03 165 PRADR1 CPY #503
F912:F0 1C F930 BEQ PRADR5
F914:06 2E 167 PRADR2 ASL FORMAT
F916:90 0E F926 BCC PRADR3
F918:BD B9 F9 169 LDX #CHAM-1,X
F91B:20 ED FD 170 JSR COUT
F91E:BD B3 F9 171 LDX #CHAM-1,X
F921:F0 03 F926 BEQ PRADR3
F923:20 ED FD 173 JSR COUT
F926:CA 174 PRADR3 DEX
F927:00 E7 F910 BNE PRADR1
F929:60 176 RTS

```

```

;FORM INDEX INTO MNEMONIC TABLE
; 1) 1XXX1010 => 00101XXX
; 2) XXX1T01 => 00111XXX
; 3) XXX1T10 => 00110XXX
; 4) XXX1T00 => 00100XXX
; 5) XXXX000 => 000XXXXXX

```

```

;GEN FMT, LEN BYTES
;SAVE MNEMONIC TABLE INDEX

```

```

;PRINT 2 BLANKS
;PRINT INST (1-3 BYTES)
;IN A 12 CTR FIELD
;CHAR COUNT FOR MNEMONIC INDEX
;RECOVER MNEMONIC INDEX

```

```

;FETCE 3-CHAR MNEMONIC
; (PACKED INTO 2-BYTES)
;SHIFT 5 BITS OF CHARACTER INTO A
; (CLEARS CARRY)
;ADD "?" OFFSET
;OUTPUT A CHAR OF MNEM
;OUTPUT 3 BLANKS
;CNT FOR 6 FORMAT BITS
;IF X=3 THEN ADDR.

```

```

F92A: 177 *
F92A:88 178 PRADR4 DEY
F92B:30 E7 F914 BHI PRADR2
F92B:30 DA FD 180 JSR PRBYTE
F930:A5 2E 181 PRADR5 LDA FORMAT
F932:C9 E8 182 CMP #SEB
F934:B1 3A 183 LDA (PC),Y
F936:90 F2 F92A BCC PRADR4
F938:20 56 F9 JSR PCADR3
F93B:AA 186 TAX
F93C:E8 187 INX
F93D:00 01 F940 RNE PRINTX
F940:98 189 INY
F941:20 DA FD 191 PRINTX JSR PRBYTE
F944:8A 192 PRINTX TXA
F945:4C DA FD 193 JMP PRBYTE
F948: 194 *
F948:A2 03 195 PRBLNK LDX #03
F94A:A9 A0 196 PRBL2 LDA #9A0
F94C:20 ED FD 197 PRBL3 JSR COUNT
F94F:CA 198 DEX
F950:00 F8 F94A RNE PRBL2
F952:60 200 RTS
F953: 201 *
F953:38 202 PCADJ SEC
F954:A5 2F 203 PCADJ2 LDA LENGTH
F956:A4 3B 204 PCADJ3 LDY PCH
F958:AA 205 TAX
F959:10 01 F95C BPL PCADJ4
F95B:88 207 DEY
F95C:65 3A 208 PCADJ4 ADC PCL
F95E:90 01 F961 BCC RTS2
F961:60 210 INY
F961:60 211 RTS2 RTS
F962: 212 *
F962: 213 ; FMT1 INSTRS
F962: 214 ; IF Y=0
F962: 215 ; IF Y=1
F962: 216 ;
F962: 217 *
F962:0F 218 FMT1 DFB $0F
F963:22 219 DFB $22
F964:FF 220 DFB $FF
F965:33 221 DFB $33
F966:CB 222 DFB $CB
F967:62 223 DFB $62
F968:FF 224 DFB $FF
F969:73 225 DFB $73
F96A:03 226 DFB $03
F96B:22 227 DFB $22
F96C:FF 228 DFB $FF
F96D:33 229 DFB $33
F96E:CB 230 DFB $CB
F96F:66 231 DFB $66
F970:FF 232 DFB $FF
F971:77 233 DFB $77
F972:0F 234 DFB $0F

```

```

F973:20 235 DFB $20
F974:FF 236 DFB $FF
F975:33 237 DFB $33
F976:CB 238 DFB $CB
F977:60 239 DFB $60
F978:FF 240 DFB $FF
F979:70 241 DFB $70
F97A:0F 242 DFB $0F
F97B:22 243 DFB $22
F97C:FF 244 DFB $FF
F97D:39 245 DFB $39
F97E:CB 246 DFB $CB
F97F:66 247 DFB $66
F980:FF 248 DFB $FF
F981:7D 249 DFB $7D
F982:0B 250 DFB $0B
F983:22 251 DFB $22
F984:FF 252 DFB $FF
F985:33 253 DFB $33
F986:CB 254 DFB $CB
F987:A6 255 DFB $A6
F988:FF 256 DFB $FF
F989:73 257 DFB $73
F98A:11 258 DFB $11
F98B:22 259 DFB $22
F98C:FF 260 DFB $FF
F98D:33 261 DFB $33
F98E:CB 262 DFB $CB
F98F:A6 263 DFB $A6
F990:FF 264 DFB $FF
F991:87 265 DFB $87
F992:01 266 DFB $01
F993:22 267 DFB $22
F994:FF 268 DFB $FF
F995:33 269 DFB $33
F996:CB 270 DFB $CB
F997:60 271 DFB $60
F998:FF 272 DFB $FF
F999:70 273 DFB $70
F99A:01 274 DFB $01
F99B:22 275 DFB $22
F99C:FF 276 DFB $FF
F99D:33 277 DFB $33
F99E:CB 278 DFB $CB
F99F:60 279 DFB $60
F9A0:FF 280 DFB $FF
F9A1:70 281 DFB $70
F9A2:24 282 DFB $24
F9A3:31 283 DFB $31
F9A4:65 284 DFB $65
F9A5:78 285 DFB $78
F9A6: 286 ; ZXXXXY0 INSTR S
F9A6:00 287 FMT2 DFB $00
F9A7:21 288 DFB $21
F9A8:81 289 DFB $81
F9A9:82 290 DFB $82
F9AA:59 291 DFB $59
F9AB:4D 292 DFB $4D

```

```

;ERR
;IMM
;E-PAGE
;ABS
;(2FAG,X)
;(2FAG),Y

```

```

F9E4:A5      351      DFB $A5
F9E5:69      352      DFB $69
F9E6:24      353      DFB $24
F9E7:24      354      DFB $24
F9E8:A2      355      DFB $A2
F9E9:A2      356      DFB $A2
F9EA:88      357      DFB $88
F9EB:AD      358      DFB $AD
F9EC:29      359      DFB $29
F9ED:8A      360      DFB $8A
F9EE:7C      361      DFB $7C
F9EF:8B      362      DFB $8B
F9F0:15      363      DFB $15
F9F1:9C      364      DFB $9C
F9F2:6D      365      DFB $6D
F9F3:9C      366      DFB $9C
F9F4:A5      367      DFB $A5
F9F5:69      368      DFB $69
F9F6:29      369      DFB $29
F9F7:53      370      DFB $53
F9F8:84      371      DFB $84
F9F9:13      372      DFB $13
F9FA:34      373      DFB $34
F9FB:11      374      DFB $11
F9FC:A5      375      DFB $A5
F9FD:69      376      DFB $69
F9FE:23      377      DFB $23
F9FF:A0      378      DFB $A0
FA00:         379 +
FA00:D8      380 MMEM
FA01:62      381      DFB $62
FA02:5A      382      DFB $5A
FA03:48      383      DFB $48
FA04:26      384      DFB $26
FA05:62      385      DFB $62
FA06:94      386      DFB $94
FA07:88      387      DFB $88
FA08:54      388      DFB $54
FA09:44      389      DFB $44
FA0A:C8      390      DFB $C8
FA0B:34      391      DFB $34
FA0C:68      392      DFB $68
FA0D:44      393      DFB $44
FA0E:E8      394      DFB $E8
FA0F:94      395      DFB $94
FA10:C4      396      DFB $C4
FA11:B4      397      DFB $B4
FA12:08      398      DFB $08
FA13:84      399      DFB $84
FA14:74      400      DFB $74
FA15:B4      401      DFB $B4
FA16:28      402      DFB $28
FA17:62      403      DFB $62
FA18:74      404      DFB $74
FA19:F4      405      DFB $F4
FA1A:CC      406      DFB $CC
FA1B:4A      407      DFB $4A
FA1C:72      408      DFB $72

```

; (B) FORMAT

; (C) FORMAT

; (D) FORMAT

; (E) FORMAT

;BRA

```

F9AC:91      293      DFB $91
F9AD:92      294      DFB $92
F9AE:86      295      DFB $86
F9AF:4A      296      DFB $4A
F9B0:85      297      DFB $85
F9B1:9D      298      DFB $9D
F9B2:49      299      DFB $49
F9B3:5A      300      DFB $5A
F9B4:         301 +
F9B4:D9      302 CHAR2
F9B5:00      303      DFB $00
F9B6:D8      304      DFB $D8
F9B7:A4      305      DFB $A4
F9B8:A4      306      DFB $A4
F9B9:00      307      DFB $00
F9BA:         308 +
F9BA:AC      309 CHAR1
F9BB:A9      310      DFB $A9
F9BC:AC      311      DFB $AC
F9BD:A3      312      DFB $A3
F9BE:A8      313      DFB $A8
F9BF:A4      314      DFB $A4
F9C0:1C      315 MMEM
F9C1:8A      316      DFB $8A
F9C2:1C      317      DFB $1C
F9C3:23      318      DFB $23
F9C4:5D      319      DFB $5D
F9C5:88      320      DFB $88
F9C6:1B      321      DFB $1B
F9C7:A1      322      DFB $A1
F9C8:9D      323      DFB $9D
F9C9:8A      324      DFB $8A
F9CA:1D      325      DFB $1D
F9CB:23      326      DFB $23
F9CC:9D      327      DFB $9D
F9CD:88      328      DFB $88
F9CE:1D      329      DFB $1D
F9CF:A1      330      DFB $A1
F9D0:1C      331      DFB $1C
F9D1:29      332      DFB $29
F9D2:19      333      DFB $19
F9D3:AE      334      DFB $AE
F9D4:69      335      DFB $69
F9D5:A6      336      DFB $A6
F9D6:19      337      DFB $19
F9D7:23      338      DFB $23
F9D8:24      339      DFB $24
F9D9:53      340      DFB $53
F9DA:1B      341      DFB $1B
F9DB:23      342      DFB $23
F9DC:24      343      DFB $24
F9DD:53      344      DFB $53
F9DE:19      345      DFB $19
F9DF:A1      346      DFB $A1
F9E0:AD      347      DFB $AD
F9E1:1A      348      DFB $1A
F9E2:5B      349      DFB $5B
F9E3:5B      350      DFB $5B

```

; (A) FORMAT ABOVE  
; TSB

21 AUTOSTI	Apple //c F8 monitor firmware	20-OCT-86 06:41 PAGE 75	21 AUTOSTI	Apple //c F8 monitor firmware	20-OCT-86 06:41 PAGE 76
FAID:F2	DFB \$F2		FA56:6C F0 03	JMP (BRKV)	
FAIE:A4	DFB \$A4		FA59:		;call BRK HANDLER
FAIF:8A	DFB \$8A	; (A) FORMAT	FA59:20 82 F8	JSR INSDI1	;PRINT USER PC
FAJ0:06	DFB \$06	; TSB	FA5C:20 DA FA	JSR IGDSP1	; AND RGS
FAJ1:5A	DFB \$5A		FA5F:4C 65 FF	JMP MON	;GO TO MONITOR (NO PASS GO, NO \$2001)
FAJ2:A2	DFB \$A2		FA62:		;DO THIS FIRST THIS TIME
FAJ3:A2	DFB \$A2		FA62:D8	473 RESET	
FAJ4:74	DFB \$74		FA63:20 84 FE	474	
FAJ5:74	DFB \$74		FA66:20 2F FB	475	
FAJ6:74	DFB \$74		FA69:20 4D CE	476	
FAJ7:72	DFB \$72	; (B) FORMAT	FA69:20 40 C7	477	
FAJ8:44	DFB \$44		FA69:20 04 CC	478	
FAJ9:68	DFB \$68		FA72:9C FC 04	479	
FAJ2A:B2	DFB \$B2		FA75:AD 5F C0	480	
FAJ2B:32	DFB \$32		FA78:20 6D FA	481	
FAJ2C:B2	DFB \$B2		FA7B:2C 10 C0	482	
FAJ2D:72	DFB \$72		FA7E:80 05 FA85	483	
FAJ2E:22	DFB \$22		FA80:8A	484	
FAJ2F:72	DFB \$72	; (C) FORMAT	FA81:D8	485 NEWMON	
FAJ3:1A	DFB \$1A		FA82:20 3A FF	486	
FAJ31:1A	DFB \$1A		FA85:AD F3 03	487 BEPSKIP	
FAJ32:26	DFB \$26		FA88:49 A5	488	
FAJ33:26	DFB \$26		FA8A:CD F4 03	489	
FAJ34:72	DFB \$72		FA8B:D0 17 FA86	490	
FAJ35:72	DFB \$72		FA8D:AD F2 03	491	
FAJ36:88	DFB \$88		FA92:D0 0F FA83	492	
FAJ37:C8	DFB \$C8		FA94:A9 E0	493	
FAJ38:C4	DFB \$C4		FA96:CD F3 03	494	
FAJ39:CA	DFB \$CA		FA99:D0 08 FA83	495	
FAJ3A:26	DFB \$26		FA9B:A0 03	496 FINSEV	
FAJ3B:48	DFB \$48		FA9D:9C F2 03	497	
FAJ3C:44	DFB \$44		FAA0:4C 00 E0	498	
FAJ3D:44	DFB \$44		FAA3:	499 *	
FAJ3E:A2	DFB \$A2		FAA3:80 F2 03	500 NOTIX	
FAJ3F:C8	DFB \$C8	; (E) FORMAT	FAA6:	501 *	
FA40:			FAA6:20 CA FC	502 PMUP	
FA40:85 45	STA \$45	;+ Trash \$45 for those who want it	FAA9:20 05	503 SETPC3	
FA42:A5 45	LDA \$45		FAA9:BD FC FA	505 SETPLP	
FA44:4C 03 C8	JMP NEWING		FAAB:9D EF 03	506 STA BRKV-1,X	
FA47:			FAB1:CA	507 DEX	
FA47:			FAB2:D0 F7 FAAB	508	
FA47:			FAB4:A9 C4	509	
FA47:			FAB6:80 5A	510	
FA47:			FAB8:	511 *	
FA47:			FAB8:	512 *	
FA47:			FAB8:	513 *	
FA47:85 44	STA MACSTAT	;save state of machine	FAB8:8A	514	
FA49:7A	PLX	;restore registers for save	FAB8:8B	515	
FA4A:FA			FABA:A5	516	
FA4B:66	PLA		FAB8:AC	517	
FA4C:			FA8C:00	518	
FA4C:28	PLP	;Note: same as OLD BREAK routine!	FA8D:	519 *	
FA4D:20 4A FF	JSR SAVE	;save reg's on BRK	FA8D:	520	
FA50:68	PIA	;including PC	FA8D:	521	
FA51:83 3A	STA PCL		FA8D:	522 *	
FA53:68	PIA		FA8D:	523 *	
FA54:85 3B	STA PCE		FA8D:	524 *	

;causes delay if key bounces  
;is reset hi  
;a funny complement of the  
;pm up byte ???  
;no so pmup  
;yes see if cold start  
;has been done yet?  
;does sev point at basic?  
;yes so reenter system  
;no so point at warm start  
;for next reset  
;and do the cold start  
;trash memory, init ports  
;set page 3 vectors  
;with ctrl b ads  
;of current basic  
;load hi slot +1  
;branch around mnemonics  
;extension to MEXL (left mnemonics)  
;PHY  
;PIY  
;STZ  
;TRB  
;???  
;this extension to the monitor reset routine (SPA62)  
;checks for apple keys. If both are pressed, it goes  
;into an exerciser mode. If the open apple key only is  
;pressed, memory is selectively trashed and a cold start  
;is done.

```

FB28:10 04 FB2E 583 BPL RTS2D
FB2A:C8 584 INY
FB2B:D0 F8 FB25 585 BNE PREAD2
FB2D:88 586 DEY
FB2E:60 587 RTS
FB2F: 61 INCLUDE AUTOST2

```

```

FAD0: 525 * LDA #FFF
FAD8:A9 FF 526 RESET.X
FAD9:8D FB 04 527 STA VMOKE
FAC2:20 3A FF 528 JSR BELL
FAC5:20 F8 C5 529 JSR PCWRST
FAC8:0E 62 C0 530 ASL BUTNI
FACB:2C 61 C0 531 BIT BUTNO
FACE:10 5E FB2E 532 BPL RTS2D
FAD0:90 D4 FFA6 533 BCC PRUP
FAD2:4C C1 C7 534 JMP BANGER
FAD5:EA 535 NOP
FAD6:EA 536 NOP
FAD7:20 8E FD 537 REGDSP JSR CROUT
FADA:A9 44 538 RGDSP1 LDA #844
FADC:85 40 539 STA A31
FADE:A9 00 540 LDA #800
FAE0:85 41 541 STA A3H
FAE2:A2 FA 542 LDX #8FA
FAE4:A9 A0 543 RDSP1 LDA #8A0
FAE6:20 ED FD 544 JSR COUT
FAE9:BD 9A CE 545 LDA RTBL-SFA,X
FADC:20 ED FD 546 JSR COUT
FAEF:A9 BD 547 LDA #8BD
FAF1:20 ED FD 548 JSR COUT
FAF4:B5 4A 549 LDA ACC+5,X
FAF6:80 0A FB02 550 BRA RGDSP2
FAF8: 551 *
FAF8: 552 * Right half of new mnemonics, indexed from MNEMR
FAF8: 553 *
FAF8: 554 DFB $74 ;PHY
FAF9:74 555 DFB $7A ;PLY
FAFA:76 556 DFB $76 ;STZ
FAFB:C6 557 DFB $C6 ;TRB
FAFC:00 558 DFB $00 ;???
FAFD: 559 *
FAFD:59 FA 560 PWRCON DW OLDRBK
FAFE:00 E0 45 561 DFB $00,$E0,$45
FB02: 562 *
FB02:20 DA FD 563 RGDSP2 JSR PRYTE
FB05:E8 564 INX
FB06:30 DC FAE4 565 BMI RDSP1
FB08:60 566 RTS
FB09: 567 *
FB09:C1 F0 EC 568 TITLE ASC 'Apple
FB11:C4 569 DFB $C4 ;optional filler
FB12: 570 *
FB12:86 00 571 PWRUP2 STX LOC0
FB14:85 01 572 STA LOC1
FB16:20 60 FB 573 JSR APPLEII
FB19:6C 00 00 574 JMP (LOC0)
FB1C:00 575 BRK
FB1D:00 576 BRK
FB1E: 577 *
FB1E:4C 00 C9 578 PREAD JMP MPADDLE
FB21:A0 00 579 LOY #800
FB23:EA 580 NOP
FB24:EA 581 NOP
FB25:BD 64 C0 582 PREAD2 LDA PADDLO,X

```

```

22 AUTOST2      Apple //c F8 monitor firmware

FB2F:          2 *      LDA #500
FB2F:A9 00     3 INIT   STA STATUS
FB31:85 48     4        LDA LORES
FB33:AD 56 C0   5        LDA LINES
FB36:AD 54 C0   6        LDA TXPAGE1
FB39:AD 51 C0   7 SETTXT LDA TXTSET
FB3C:A9 00     8        LDA #500
FB3E:F0 0B     9        BEQ SETWMD
FB40:AD 50 C0  10 SETGR LDA TXCLR
FB43:AD 53 C0  11        LDA MIXSET
FB46:20 36 F8  12        JSR CRTOP
FB49:A9 14     13        LDA #514
FB4B:85 22     14 SETWMD STA WNDTOP
FB4D:EA       15        NOP
FB4E:EA       16        NOP
FB4F:20 0A CE  17        JSR WNDREST
FB52:80 05     18        BRA VTAB23
FB54:         19 *
FB54:09 80     20 DOCTL ORA #580
FB56:4C 54 CD  21        JMP CTRCHAR0
FB59:         22 *
FB59:A9 17     23 VTAB23 LDA #517
FB5B:85 25     24 TABV  STA CV
FB5D:4C 22 FC  25        JMP VTAB
FB60:         26 *
FB60:20 58 FC  27 APPLI11 JSR HOME
FB63:A0 09     28        LDA #9
FB65:B9 BA C5  29 STITLE LDA APPLE2C-1,Y
FB68:99 0D 04  30        STA LINE1+13,Y
FB6A:88       31        DEY
FB6C:D0 F7     32        BNE STITLE
FB6E:60       33        RTS
FB6F:         34 *
FB6F:AD E3 03  35 SETPHRC LDA SORTEV+1
FB72:A9 A5     36        EOR #5A5
FB74:8D F4 03  37        STA PHREDUP
FB77:60       38
FB78:         39 *
FB78:      FB78 40 VIDWAIT RTS
FB78:C9 8D     41        CMP #58D
FB7A:D0 18     42        BNE NOWAIT
FB7C:AC 00 C0   43        LDY KED
FB7F:10 13     44        BPL NOWAIT
FB81:C0 93     45        CPY #593
FB83:D0 0F     46        BEQ NOWAIT
FB85:2C 10 C0   47        BIT KEDSTRB
FB88:AC 00 C0   48        KEDWAIT LDY KED
FB8B:10 F8     49        BPL KEDWAIT
FB8D:C0 83     50        CPY #583
FB8F:F0 03     51        BEQ NOWAIT
FB91:2C 10 C0   52        BIT KEDSTRB
FB94:2C 7B 06   53 NOWAIT BIT VTRACTV
FB97:30 64     54        BPL VIDOUT
FB99:89 60     55        BIT #560
FB9B:F0 B7     56        BEQ DOCTL
FB9D:20 B8 C3   57        JSR STORCH
FB9E:EE 7B 05   58 REMADV INC OURCH
FB9F:AD 7B 05   59        LDA

```

```

20-OCT-86 06:41 PAGE 79

;CLR STATUS FOR DEBUG SOFTWARE

;INIT VIDEO MODE
;SET FOR TEXT MODE
;FULL SCREEN WINDOW

;SET FOR GRAPHICS MODE
;LOWER 4 LINES AS TEXT WINDOW

;SET WINDOW

;40/80 column width

;controls need high bit
;execute control char

;VTAB TO ROW 23
;VTABS TO ROW IN A-REG
;don't set OURCV!!

;CLEAR THE SCRN

;GET A CHAR
;PUT IT AT TOP CENTER OF SCREEN

;ROUTINE TO CALCULATE THE 'FUNNY
;COMPLEMENT' FOR THE RESET VECTOR

;CHECK FOR A PAUSE (CONTROL-S).
;ONLY WHEN I HAVE A CR
;NOT SO, DO REGULAR
;IS KEY PRESSED?
;NO.
;YES -- IS IT CTRL-S?
;NOPE - IGNORE
;CLEAR STROBE
;WAIT TILL NEXT KEY TO RESUME
;WAIT FOR KEYPRESS
;IS IT CONTROL-C?
;YES, SO LEAVE IT
;CLR STROBE
;is video firmware active?
;=>no, do normal 40 column
;is it a control?
;=>yes, do it
DOCTL
;advance cursor
;and update others

```

```

22 AUTOST2      Apple //c F8 monitor firmware

FB96:2C 1F C0   60        FB9C:30 05     61        BIT WNDREVID
FB9A:30 05     62        BMI NEMADV1
FB9B:8D 7B 04   63        STA OLDCH
FB9E:85 24     64        BRA ADV2
FB9F:80 46     65 *
FB9F:EA       66        NOP
FB9F:EA       67 *
FB9F:85 24     68 REVERSE10W DFB GOODF8
FB9F:85 24     69 *
FB9F:10 06     70 DOCCUT1 BPL DCX
FB9F:C9 A0     71        CMP #5A0
FB9F:90 02     72        BCC DCX
FB9F:25 32     73        AND INVTLG
FB9F:4C F6 FD   74        DCK JMP COUNT
FB9F:03       75        ddb #63
FB9F:01       76 *
FB9F:00       77        DFB $00
FB9F:18       78 *
FB9F:18       79 BASCALC PBA
FB9F:48       80        LSR A
FB9F:23 03     81        AND #503
FB9F:09 04     82        ORA #504
FB9F:85 29     83        STA BASH
FB9F:68       84        PLA
FB9F:23 18     85        AND #518
FB9F:90 02     86        BCC BASCLC2
FB9F:69 7F     87        ADC #57F
FB9F:85 28     88 BASCLC2 STA BASL
FB9F:0A       89        ASL A
FB9F:0A       90        ASL A
FB9F:05 28     91        ORA BASL
FB9F:85 28     92        STA BASL
FB9F:60       93        RTS
FB9F:9C       94 *
FB9F:C9 87     95 CHKBELL CMP #587
FB9D:D0 12     96        BNE RTS2B
FB9D:A9 40     97 BELL1 LDA #540
FB9D:20 A8 FC   98        LDY #5C0
FB9E:A9 0C     99        LDA #50C
FB9E:20 A8 FC  100 BELL2 LDA #50C
FB9E:AD 30 C0  101        LDA SPBR
FB9E:86       102        DEY
FB9E:D0 F5     103        BNE BELL2
FB9E:60       104        RTS
FB9E:60       105 RTS2B
FB9E:60       106 *
FB9E:A4 24     107 STORADV STA (BASI),Y
FB9E:91 28     108        LDY #5C0
FB9E:26 24     109 ADVANCE INC CH
FB9E:A5 24     110        LDA CH
FB9E:C5 21     111 ADV2  CMP WNDMOTH
FB9E:B0 66     112        BCS CR
FB9E:60       113 RTS3  RTS
FB9E:60       114 *
FB9E:60       115 VIDOUT CMP #5A0
FB9E:C9 A0     116        BPL FB9F:8D EF
FB9F:8D EF     117        TAY

```

```

20-OCT-86 06:41 PAGE 80

;but only if not 80 columns
;=>80 columns, leave em

;check for CR

;e, chels ID byte

;=>video firmware active, no mask
;is it control char?
;=>yes, no mask
;else apply inverse mask
;and print character
;revision byte

;chels ID byte

;CALC BASE ADDR IN BASI,H
;FOR GIVEN LINE NO.
;0<=LINE NO<=517
;ARG=000ARCODE, GENERATE
;BASE=00000ICD
;AND
;BASL=2ARAR000

;BELL CHAR? (CONTROL-G)
; NO, RETURN.
; YES...
;DELAY .01 SECONDS
;TOGGLE SPANER AT 1 KHZ
; FOR .1 SEC.

;get 40 column position
;and store
;increment cursor
;BEYOND WINDOW WIDTH?
; YES, CR TO NEXT LINE.
; NO, RETURN.
;CONTROL CHAR?
; NO, OUTPUT IT.
;INVERSE VIDEO?

```

```

FC02:10 EC FB70 118 BPL STORADV ; YES, OUTPUT IT.
FC04:C9 8D FB73 120 BEQ #80 ;CR?
FC06:F0 6B FB73 120 BEQ #80 ;Yes, use new routine
FC08:C9 8A FB73 121 CMP #8A ;LINE FED?
FC0A:F0 5A FC66 122 BEQ LF ; IF 80, DO IT.
FC0C:C9 88 FB73 123 CMP #98 ;BACK SPACE? (CONTROL-R)
FC0E:D0 C9 FED9 124 BNE CHKBELL ; NO, CHECK FOR BELL.
FC10:20 E2 FE 125 BS JSR DECC ;decrement all cursor H indices
FC11:10 E7 FB7C 126 RPL RTS ;IF POSITIVE, OK; ELSE MOVE UP.
FC13:A5 21 FB7C 127 LDA WNDWTH ;get window width.
FC15:20 E8 FE 128 JSR WDRCHR ;and set CH's to WNDWTH-1
FC17:A5 22 FC1A:55 22 LDA WNDWTH ;CURSOR V INDEX
FC19:C5 25 FC1C:65 25 CMP CV ;top line, exit
FC1B:80 DC FB7C 131 BCS RTS ;not top, go up one
FC1D:C6 25 FC22: 132 DEC CV
FC24: 133 *
FC26:80 62 FC66 134 VTBZ BRA NEWTAB
FC28:20 C1 FB 135 VTBZ JSR BASCALC
FC2A:55 20 FC2B:2C 1F C0 136 LDA WNDLFT ;calculate the base address
FC2C:2C 1F C0 137 BIT RDRVID ;get the left window edge
FC2E:10 02 FC30 138 BPL VTB40 ;80 columns?
FC2F:18 139 JSR A ;>no, left edge ok
FC31: 140 CLC ;divide width by 2
FC33: 141 VTB40 ADC BASL ;prepare to add
FC35: 142 STA BASL ;add width to base
FC37:85 28 FC3A:60 RTS
FC39: 143 RTS4
FC41: 144 *
FC43: 145 *
FC45: 146 *
FC47: 147 *
FC49: 148 *
FC51: 149 *
FC53: 150 *
FC55: 151 *
FC57: 152 *
FC59: 153 *
FC61: 154 *
FC63: 155 *
FC65: 156 *
FC67: 157 *
FC69: 158 *
FC71: 159 *
FC73: 160 *
FC75: 161 *
FC77: 162 *
FC79: 163 *
FC81: 164 *
FC83: 165 *
FC85: 166 *
FC87: 167 *
FC89: 168 *
FC91: 169 *
FC93: 170 *
FC95: 171 *
FC97: 172 *
FC99: 173 *
FC101: 174 *
FC103: 175 *
FC105: 176 *
FC107: 177 *
FC109: 178 *
FC111: 179 *
FC113: 180 *
FC115: 181 *
FC117: 182 *
FC119: 183 *
FC121: 184 *
FC123: 185 *
FC125: 186 *
FC127: 187 *
FC129: 188 *
FC131: 189 *
FC133: 190 *
FC135: 191 *
FC137: 192 *
FC139: 193 *
FC141: 194 *
FC143: 195 *
FC145: 196 *
FC147: 197 *
FC149: 198 *
FC151: 199 *
FC153: 200 *
FC155: 201 *
FC157: 202 *
FC159: 203 *
FC161: 204 *
FC163: 205 *
FC165: 206 *
FC167: 207 *
FC169: 208 *
FC171: 209 *
FC173: 210 *
FC175: 211 *
FC177: 212 *
FC179: 213 *
FC181: 214 *
FC183: 215 *
FC185: 216 *
FC187: 217 *
FC189: 218 *
FC191: 219 *
FC193: 220 *
FC195: 221 *
FC197: 222 *
FC199: 223 *
FC201: 224 *
FC203: 225 *
FC205: 226 *
FC207: 227 *
FC209: 228 *
FC211: 229 *
FC213: 230 *
FC215: 231 *
FC217: 232 *
FC219: 233 *
FC221: 234 *
FC223: 235 *
FC225: 236 *
FC227: 237 *
FC229: 238 *
FC231: 239 *
FC233: 240 *
FC235: 241 *
FC237: 242 *
FC239: 243 *
FC241: 244 *
FC243: 245 *
FC245: 246 *
FC247: 247 *
FC249: 248 *
FC251: 249 *
FC253: 250 *
FC255: 251 *
FC257: 252 *
FC259: 253 *
FC261: 254 *
FC263: 255 *
FC265: 256 *
FC267: 257 *
FC269: 258 *
FC271: 259 *
FC273: 260 *
FC275: 261 *
FC277: 262 *
FC279: 263 *
FC281: 264 *
FC283: 265 *
FC285: 266 *
FC287: 267 *
FC289: 268 *
FC291: 269 *
FC293: 270 *
FC295: 271 *
FC297: 272 *
FC299: 273 *
FC301: 274 *
FC303: 275 *
FC305: 276 *
FC307: 277 *
FC309: 278 *
FC311: 279 *
FC313: 280 *
FC315: 281 *
FC317: 282 *
FC319: 283 *
FC321: 284 *
FC323: 285 *
FC325: 286 *
FC327: 287 *
FC329: 288 *
FC331: 289 *
FC333: 290 *
FC335: 291 *
FC337: 292 *
FC339: 293 *
FC341: 294 *
FC343: 295 *
FC345: 296 *
FC347: 297 *
FC349: 298 *
FC351: 299 *
FC353: 300 *
FC355: 301 *
FC357: 302 *
FC359: 303 *
FC361: 304 *
FC363: 305 *
FC365: 306 *
FC367: 307 *
FC369: 308 *
FC371: 309 *
FC373: 310 *
FC375: 311 *
FC377: 312 *
FC379: 313 *
FC381: 314 *
FC383: 315 *
FC385: 316 *
FC387: 317 *
FC389: 318 *
FC391: 319 *
FC393: 320 *
FC395: 321 *
FC397: 322 *
FC399: 323 *
FC401: 324 *
FC403: 325 *
FC405: 326 *
FC407: 327 *
FC409: 328 *
FC411: 329 *
FC413: 330 *
FC415: 331 *
FC417: 332 *
FC419: 333 *
FC421: 334 *
FC423: 335 *
FC425: 336 *
FC427: 337 *
FC429: 338 *
FC431: 339 *
FC433: 340 *
FC435: 341 *
FC437: 342 *
FC439: 343 *
FC441: 344 *
FC443: 345 *
FC445: 346 *
FC447: 347 *
FC449: 348 *
FC451: 349 *
FC453: 350 *
FC455: 351 *
FC457: 352 *
FC459: 353 *
FC461: 354 *
FC463: 355 *
FC465: 356 *
FC467: 357 *
FC469: 358 *
FC471: 359 *
FC473: 360 *
FC475: 361 *
FC477: 362 *
FC479: 363 *
FC481: 364 *
FC483: 365 *
FC485: 366 *
FC487: 367 *
FC489: 368 *
FC491: 369 *
FC493: 370 *
FC495: 371 *
FC497: 372 *
FC499: 373 *
FC501: 374 *
FC503: 375 *
FC505: 376 *
FC507: 377 *
FC509: 378 *
FC511: 379 *
FC513: 380 *
FC515: 381 *
FC517: 382 *
FC519: 383 *
FC521: 384 *
FC523: 385 *
FC525: 386 *
FC527: 387 *
FC529: 388 *
FC531: 389 *
FC533: 390 *
FC535: 391 *
FC537: 392 *
FC539: 393 *
FC541: 394 *
FC543: 395 *
FC545: 396 *
FC547: 397 *
FC549: 398 *
FC551: 399 *
FC553: 400 *
FC555: 401 *
FC557: 402 *
FC559: 403 *
FC561: 404 *
FC563: 405 *
FC565: 406 *
FC567: 407 *
FC569: 408 *
FC571: 409 *
FC573: 410 *
FC575: 411 *
FC577: 412 *
FC579: 413 *
FC581: 414 *
FC583: 415 *
FC585: 416 *
FC587: 417 *
FC589: 418 *
FC591: 419 *
FC593: 420 *
FC595: 421 *
FC597: 422 *
FC599: 423 *
FC601: 424 *
FC603: 425 *
FC605: 426 *
FC607: 427 *
FC609: 428 *
FC611: 429 *
FC613: 430 *
FC615: 431 *
FC617: 432 *
FC619: 433 *
FC621: 434 *
FC623: 435 *
FC625: 436 *
FC627: 437 *
FC629: 438 *
FC631: 439 *
FC633: 440 *
FC635: 441 *
FC637: 442 *
FC639: 443 *
FC641: 444 *
FC643: 445 *
FC645: 446 *
FC647: 447 *
FC649: 448 *
FC651: 449 *
FC653: 450 *
FC655: 451 *
FC657: 452 *
FC659: 453 *
FC661: 454 *
FC663: 455 *
FC665: 456 *
FC667: 457 *
FC669: 458 *
FC671: 459 *
FC673: 460 *
FC675: 461 *
FC677: 462 *
FC679: 463 *
FC681: 464 *
FC683: 465 *
FC685: 466 *
FC687: 467 *
FC689: 468 *
FC691: 469 *
FC693: 470 *
FC695: 471 *
FC697: 472 *
FC699: 473 *
FC701: 474 *
FC703: 475 *
FC705: 476 *
FC707: 477 *
FC709: 478 *
FC711: 479 *
FC713: 480 *
FC715: 481 *
FC717: 482 *
FC719: 483 *
FC721: 484 *
FC723: 485 *
FC725: 486 *
FC727: 487 *
FC729: 488 *
FC731: 489 *
FC733: 490 *
FC735: 491 *
FC737: 492 *
FC739: 493 *
FC741: 494 *
FC743: 495 *
FC745: 496 *
FC747: 497 *
FC749: 498 *
FC751: 499 *
FC753: 500 *
FC755: 501 *
FC757: 502 *
FC759: 503 *
FC761: 504 *
FC763: 505 *
FC765: 506 *
FC767: 507 *
FC769: 508 *
FC771: 509 *
FC773: 510 *
FC775: 511 *
FC777: 512 *
FC779: 513 *
FC781: 514 *
FC783: 515 *
FC785: 516 *
FC787: 517 *
FC789: 518 *
FC791: 519 *
FC793: 520 *
FC795: 521 *
FC797: 522 *
FC799: 523 *
FC801: 524 *
FC803: 525 *
FC805: 526 *
FC807: 527 *
FC809: 528 *
FC811: 529 *
FC813: 530 *
FC815: 531 *
FC817: 532 *
FC819: 533 *
FC821: 534 *
FC823: 535 *
FC825: 536 *
FC827: 537 *
FC829: 538 *
FC831: 539 *
FC833: 540 *
FC835: 541 *
FC837: 542 *
FC839: 543 *
FC841: 544 *
FC843: 545 *
FC845: 546 *
FC847: 547 *
FC849: 548 *
FC851: 549 *
FC853: 550 *
FC855: 551 *
FC857: 552 *
FC859: 553 *
FC861: 554 *
FC863: 555 *
FC865: 556 *
FC867: 557 *
FC869: 558 *
FC871: 559 *
FC873: 560 *
FC875: 561 *
FC877: 562 *
FC879: 563 *
FC881: 564 *
FC883: 565 *
FC885: 566 *
FC887: 567 *
FC889: 568 *
FC891: 569 *
FC893: 570 *
FC895: 571 *
FC897: 572 *
FC899: 573 *
FC901: 574 *
FC903: 575 *
FC905: 576 *
FC907: 577 *
FC909: 578 *
FC911: 579 *
FC913: 580 *
FC915: 581 *
FC917: 582 *
FC919: 583 *
FC921: 584 *
FC923: 585 *
FC925: 586 *
FC927: 587 *
FC929: 588 *
FC931: 589 *
FC933: 590 *
FC935: 591 *
FC937: 592 *
FC939: 593 *
FC941: 594 *
FC943: 595 *
FC945: 596 *
FC947: 597 *
FC949: 598 *
FC951: 599 *
FC953: 600 *
FC955: 601 *
FC957: 602 *
FC959: 603 *
FC961: 604 *
FC963: 605 *
FC965: 606 *
FC967: 607 *
FC969: 608 *
FC971: 609 *
FC973: 610 *
FC975: 611 *
FC977: 612 *
FC979: 613 *
FC981: 614 *
FC983: 615 *
FC985: 616 *
FC987: 617 *
FC989: 618 *
FC991: 619 *
FC993: 620 *
FC995: 621 *
FC997: 622 *
FC999: 623 *

```

Apple //c P8 monitor firmware	20-OCT-86	06:41	PAGE 83
234	CMF AZL		; AND COMPARE TO A2
235	LOA A1H		; (CARRY SET IF >=)
236	SBC A2H		
237	INC A1L		
238	BNE RTS4B		
239	INC A1H		
240	RTS4B		
241	*		
242	HEADR		;don't do it
243	RTS		
244	*		
245	COLDSTART LDY		;let it precess down
246	STZ A1L		
247	LOX		;start from BFX down
248	STX A1H		
249	LOA		;store blanks
250	STA (A1L),Y		
251	STA (A1L),Y		
252	DEX		;back down to next page
253	CPX #1		;stay away from stack
254	BNE BLAST		;fall into COMINIT
255	*		
256	STA SET80COL		;init ALT screen holes
257	LOA TXTPAGE2		;for serial and comm ports
258	LDX		;C = 1 from CPX #1
259	COM1		;XFER from rom
260	BCC COM2		;branch if defaults ok
261	CPM		;test for prior setup
262	CLC		;branch if not valid
263	BNE COM2		;if 44F8 & 44FF = TEL values
264	CPX		
265	BCC COM3		
266	STA		;move all 8...
267	DEX		
268	BNE COM1		
269	COM3		;restore switches
270	STA		;to default states
271	RTS		
272	NOP		;+
273	NOP		
274	NOP		
275	NOP		
276	NOP		
277	NOP		
278	NOP		
279	NOP		
280	NOP		
281	NOP		
282	*		
283	RNKEY		;get char at current position
284	LOA		;for those who restore it
285	NOP		;if a program controls input
286	NOP		;hooks, no cursor may be displayed
287	NOP		
288	NOP		
289	NOP		
290	NOP		
291	NOP		

Apple //c F8 monitor firmware	20-OCT-86	06:41	PAGE 84
222	FD17:EA	NOP	
293 *			
294	FD18:6C 38 00	JMP	;GO TO USER KEY-IN
295 *			
296	KEYIN	STA	;erase false images
297	FD1B:91 28	SHOWCR	;display true cursor
298	DOWNTCUR	JSR	UPDNME
299	FD20:20 70 CC	BFL	;loop until keypress
300	FD25:48	PHA	;save character
301	FD26:A9 08	LDA	M.CTL
302	FD28:2C FB 04	BIT	WMOE
303	FD2B:30 07	BNE	NOESC2
304	FD2D:68	PLA	
305	FD2E:C9 98	*ESC	
306	FD30:00 06	BNE	LOOKPICK
307	FD32:4C CC	JMP	NEWESC
308 *			
309	FD35:4C ED CC	JMP	ESCDREXY
310 *			
311	FD38:2C 78 06	BIT	LOOKPICK
312	FD3B:30 07	BMI	NOESCAPE
313	FD3D:C9 95	CMF	*PICK
314	FD3F:00 03	BNE	NOESCAPE
315	FD41:20 1D CC	JSR	PICKY
316 *			
317	FD44:		;NOESCAPE is used by GETCOUT too.
318 *			
319	NOESCAPE	PHA	;save it
320	NOESC1	LDA	M.CTL
321	TSB	WMOE	;disable escape sequences
322	NOESC2	PLA	;and enable controls
323	RTS		;by setting M.CTL
324 *			
325	NOESC	NOP	
326 *			
327	NOTCR	JSR	GETCOUT
328	FD50:C9 88	CMF	*\$88
329	FD52:F0 1D	BEQ	BCKSPC
330	FD54:C9 98	CMF	*\$98
331	FD56:F0 0A	BEQ	CANCEL
332	FD58:E0 F8	CPX	*\$F8
333	FD5A:90 03	BCC	NOTCR1
334	FD5C:20 3A FF	JSR	BELL
335	NOTCR1	INX	
336	FD60:00 13	BNE	NTCHAR
337	FD62:A9 DC	LDA	*\$DC
338	FD64:20 A6 C3	JSR	GETCOUT
339	FD67:20 8E FD	JSR	CHOUT
340	FD6A:A5 33	LDA	PROMPT
341	FD6C:20 ED FD	JSR	COVT
342	FD6F:32 01	LDX	*\$01
343	FD71:8A	TXA	
344	FD72:F0 F3	BEQ	GETLN2
345	FD74:CA	DEX	
346	FD75:20 ED CC	BNE	NTCHAR
347	FD78:C9 95	CMF	*PICK
348	FD7A:00 08	BNE	ADDLN2
349	FD7C:20 1D CC	JSR	PICKY
			;do new ROCH (allow escapes)
			;FOR SCREEN CHAR
			;FOR CONTROL-U
			;lift char from screen

```

350 FDB3:EA NOP
351 FDB0:EA NOP
352 FDB1:EA NOP
353 FDB2:EA NOP
354 FDB3:EA NOP
355 FDB4:90 00 02 STA IN,X
356 FDB7:C9 8D CMP #8D
357 FDB9:D0 C2 FDB0 NOTCR
358 FDB8:20 9C FC FDB3 CROUT1
359 FDBE:A9 8D LDA #8D
360 FDB9:D0 5B FDB0 BNE COUT
361 FDB2:361 *
362 FDB2:A4 3D LDY A1H
363 FDB4:A6 3C LDY A1L
364 FDB6:20 8E FD FDB3 CROUT
365 FDB9:20 40 F9 FDB3 PRNTYX
366 FDB9:C0 00 LDY #500
367 FDB9:A9 AD LDA #5AD
368 FDB0:4C FD JMP COUT
369 FDB3:369 *
370 FDB3:A5 3C LDA A1L
371 FDB5:09 07 ORA #507
372 FDB7:85 3E STA A2L
373 FDB9:A5 3D LDA A1H
374 FDB8:85 3F STA A2H
375 FDB0:A5 3C LDA A1L
376 FDB7:29 07 AND #507
377 FDB1:D0 03 FDB6 BNE DATROUT
378 FDB3:20 92 FD FDB3 JSR PRA1
379 FDB6:A9 AD LDA #5A0
380 FDB8:20 ED FD FDB3 JSR COUT
381 FDB8:B1 3C FDB3 LDA (A1L),Y
382 FDB0:20 DA FD FDB3 JSR PRBYTE
383 FDB0:20 BA FC FDB3 JSR NTRAI
384 FDB3:90 E8 FDB0 BCC MOD8CHK
385 FDB3:60 RTS
386 FDB6:386 *
387 FDB6:4A LSR A
388 FDB7:90 EA FDB3 BCC XAM
389 FDB9:4A LSR A
390 FDBA:4A LSR A
391 FDB3:A5 3E FDB3 LDA A2L
392 FDBD:90 02 FDB1 BCC ADD
393 FDBF:49 FF FDB3 EOR #5FF
394 FDB1:65 3C FDB3 ADC A1L
395 FDB3:48 FDB3 PIA
396 FDB4:A9 BD LDA #8D
397 FDB6:20 ED FD FDB3 JSR COUT
398 FDB9:68 FDB3 PIA
399 FDBA:399 *
400 FDBA:48 FDBA 400 PRBYTE
401 FDB8:4A FDB3 LSR A
402 FDBC:4A FDB3 LSR A
403 FDBD:4A FDB3 LSR A
404 FDBE:4A FDB3 LSR A
405 FDBF:20 E5 FD FDB3 JSR PRHEXZ
406 FDB2:68 FDB3 PIA
407 FDB3:407 *

FDB3:29 0F FDB3 PREHEX AND #50F
FDB5:09 B0 FDB3 ORA #5B0
FDB7:C9 BA 410 CMP #5BA
FDB9:90 02 FDB0 BCC COUT
FDBB:69 06 FDB3 ADC #506
FDBD:413 *
FDBD:6C 36 00 FDB3 JMP (CSHL),Y
FDBF:415 *
FDBF:7C 78 06 FDB3 BIT VFACTV
FDBF:4C B4 FB FDB3 JMP DCCOUT1
FDBF:90 02 FDB0 STI YSAV1
FDBF:48 419 FDB3 PHA
FDBF:20 78 FB FDB3 JSR VIDWAIT
FDBF:68 421 FDB3 PLA
FDBF:44 35 FDB3 LDY YSAV1
FDBF:60 423 FDB3 RTS
FDBF:66 424 *
FDBF:C6 34 FDB3 DEC YSAV
FDBF:F0 9F FDB3 BEQ XAM8
FDBF:477 *
FDBF:CA 428 FDB3 DEX
FDBF:00 16 FDB3 BNE SETMOD2
FDBF:C9 BA 430 CMP #5BA
FDBF:00 BB FDB3 BNE XAMPM
FDBF:85 31 FDB3 STA MODE
FDBF:A5 3E FDB3 LDA A2L
FDBF:91 40 FDB3 STA (A3L),Y
FDBF:B6 40 FDB3 INC A3L
FDBF:00 02 FDB3 BNE RTS5
FDBF:86 41 FDB3 INC A3H
FDBF:60 439 FDB3 RTS
FDBF:440 *
FDBF:A4 34 FDB3 LDA IN-1,Y
FDBF:B9 FF 01 FDB3 STA SETMODE
FDBF:85 31 FDB3 LDA IN-1,Y
FDBF:60 444 FDB3 RTS
FDBF:60 445 *
FDBF:20 A2 01 FDB3 LDY #501
FDBF:85 3E FDB3 LDA A2L,X
FDBF:95 42 FDB3 STA A4L,X
FDBF:95 44 FDB3 STA A5L,X
FDBF:CA FDB3 DEX
FDBF:10 F7 FDB3 BPL LT2
FDBF:60 452 FDB3 RTS
FDBF:60 453 *
FDBF:C3 3C FDB3 LDA (A1L),Y
FDBF:91 42 FDB3 STA (A4L),Y
FDBF:20 B4 FC FDB3 JSR NTR4
FDBF:90 F7 FDB3 BCC MOVE
FDBF:60 458 FDB3 RTS
FDBF:60 459 *
FDBF:C3 3C FDB3 LDA (A1L),Y
FDBF:D1 42 FDB3 CMP (A4L),Y
FDBF:01 42 FDB3 BEQ VYOK
FDBF:20 92 FD FDB3 JSR PRA1
FDBF:B1 3C FDB3 LDA (A1L),Y
FDBF:20 DA FD FDB3 JSR PRBYTE

FDB3:29 0F FDB3 PREHEX AND #50F
FDB5:09 B0 FDB3 ORA #5B0
FDB7:C9 BA 410 CMP #5BA
FDB9:90 02 FDB0 BCC COUT
FDBB:69 06 FDB3 ADC #506
FDBD:413 *
FDBD:6C 36 00 FDB3 JMP (CSHL),Y
FDBF:415 *
FDBF:7C 78 06 FDB3 BIT VFACTV
FDBF:4C B4 FB FDB3 JMP DCCOUT1
FDBF:90 02 FDB0 STI YSAV1
FDBF:48 419 FDB3 PHA
FDBF:20 78 FB FDB3 JSR VIDWAIT
FDBF:68 421 FDB3 PLA
FDBF:44 35 FDB3 LDY YSAV1
FDBF:60 423 FDB3 RTS
FDBF:66 424 *
FDBF:C6 34 FDB3 DEC YSAV
FDBF:F0 9F FDB3 BEQ XAM8
FDBF:477 *
FDBF:CA 428 FDB3 DEX
FDBF:00 16 FDB3 BNE SETMOD2
FDBF:C9 BA 430 CMP #5BA
FDBF:00 BB FDB3 BNE XAMPM
FDBF:85 31 FDB3 STA MODE
FDBF:A5 3E FDB3 LDA A2L
FDBF:91 40 FDB3 STA (A3L),Y
FDBF:B6 40 FDB3 INC A3L
FDBF:00 02 FDB3 BNE RTS5
FDBF:86 41 FDB3 INC A3H
FDBF:60 439 FDB3 RTS
FDBF:440 *
FDBF:A4 34 FDB3 LDA IN-1,Y
FDBF:B9 FF 01 FDB3 STA SETMODE
FDBF:85 31 FDB3 LDA IN-1,Y
FDBF:60 444 FDB3 RTS
FDBF:60 445 *
FDBF:20 A2 01 FDB3 LDY #501
FDBF:85 3E FDB3 LDA A2L,X
FDBF:95 42 FDB3 STA A4L,X
FDBF:95 44 FDB3 STA A5L,X
FDBF:CA FDB3 DEX
FDBF:10 F7 FDB3 BPL LT2
FDBF:60 452 FDB3 RTS
FDBF:60 453 *
FDBF:C3 3C FDB3 LDA (A1L),Y
FDBF:91 42 FDB3 STA (A4L),Y
FDBF:20 B4 FC FDB3 JSR NTR4
FDBF:90 F7 FDB3 BCC MOVE
FDBF:60 458 FDB3 RTS
FDBF:60 459 *
FDBF:C3 3C FDB3 LDA (A1L),Y
FDBF:D1 42 FDB3 CMP (A4L),Y
FDBF:01 42 FDB3 BEQ VYOK
FDBF:20 92 FD FDB3 JSR PRA1
FDBF:B1 3C FDB3 LDA (A1L),Y
FDBF:20 DA FD FDB3 JSR PRBYTE

```

```

22 AUTOST2      Apple //c P8 monitor firmware
FE44:A9 A0      466      LDA #SA0
FE46:20 ED FD   467      JSR COUNT
FE49:A9 A8      468      LDA #SA8
FE4B:20 ED FD   469      JSR COUNT
FE4E:B1 42      470      LDA (AAL),Y
FE50:20 DA FD   471      JSR PRBTE
FE53:A9 A9      472      LDA #SA9
FE55:20 ED FD   473      JSR COUNT
FE58:20 B4 FC   474      JSR NUTAA
FE5B:90 D9      475      BCC VERIFY
FE5D:60         476      RTS
FE5E:         477      *
FE5E:20 75 FE   478      JSR AIPC
FE61:A3 14      479      LDA #S14
FE63:48         480      PBA
FE64:20 C4 C5   481      JSR SROWINST
FE67:68         482      PLA
FE68:3A         483      DEC A
FE69:D0 F8      484      BNE LIST2
FE6B:60         485      RTS
FE6C:         486      *
FE6C:4C 86 C9   487      JMP GETINST1
FE6F:C6 34      488      DEC TRACE
FE71:4C 43 CA   489      JMP STEP
FE74:         0001 490      ds $FE75-*,0
FE75:         491      *
FE75:8A         492      TAA
FE76:F0 07      493      BQ AIPCRYS
FE78:B5 3C      494      LDA AIL,X
FE7A:95 3A      495      STA PCL,X
FE7C:CA         496      DEX
FE7D:10 F9      497      BPL AIPCLP
FE7F:60         498      RTS
FE80:         499      *
FE80:A0 3F      500      SETINV
FE82:D0 02      501      BNE SETIFLG
FE84:A0 FF      502      SETNORM
FE86:84 32      503      SETIFLG
FE88:60         504      RTS
FE89:         505      *
FE89:A9 00      506      SETKBD
FE8B:85 3E      507      JMPRT
FE8D:A2 38      508      INPR
FE8F:A0 1B      509      LDY
FE91:D0 08      509      BNE IOPRT
FE93:         511      *
FE93:A9 00      512      SETVID
FE95:83 3E      513      OUTPORT
FE97:A2 36      514      OUTPT
FE99:A0 F0      515      LDY
FE9B:A5 3E      516      LDA A2L
FE9D:29 0F      517      AND #50F
FE9F:D0 06      518      BNE NOTPRTO
FEA1:C0 1B      519      CPY #KEYIN
FEA3:F0 39      520      BQ IOPRT1
FEA5:80 1B      521      BRA OPRT0
FEA7:09 C0      522      NOTPRTO
FEA9:A0 00      523      LDY

```

```

20-OCT-86 06:41 PAGE 87
;MOVE A1 (2 BYTES) TO
; PC IF SPEC'D AND
;+DISASSEMBLE 20 INSTRUCTIONS.
;+display a line
;+Count down
;+Go to the mini assembler
;+Stay on T for trace
;+off to the step routine
;+Extra bytes
;+IF USER SPECIFIED AN ADDRESS,
; COPY IT FROM A1 TO PC.
;+YES, SO COPY IT.
;SET FOR INVERSE VID
; VIA COUNT1
;SET FOR NORMAL VID
;DO 'IN#0'
;DO 'IN#ARE'
;DO 'PR#0'
;DO 'PR#ARE'
;not slot 0
;Continue if KEYIN
;=>do PR#0

```

```

22 AUTOST2      Apple //c P8 monitor firmware
FEAB:94 00      524      IOPRT2
FEAD:95 01      525      STA LOC1,X
FEAF:60         526      RTS
FEB0:         527      *
FEB0:4C 00 E0   528      XBASIC
FEB3:         529      *
FEB3:4C 03 E0   530      BASCONT
FEB6:         531      *
FEB6:20 75 FE   532      JSR AIPC
FEB9:20 3F FF   533      JSR RESTORE
FEBC:6C 3A 00   534      JMP (PCL)
FEBD:         535      *
FEBD:4C D7 FA   536      JMP REGDSP
FECE:         537      *
FECE:3A         538      OPRT0
FECD:8D 7B 06   539      STA CURSOR
FECE:A3 F7      540      LDA #SFF-M.CTL
FECE:80 04      541      BRA DOPR0
FECA:         542      *
FECA:4C F8 03   543      JMP USRADR
FECD:         544      *
FECD:60         545      WRITE
FECE:         546      *
FECE:8D 7B 06   547      DOPR0
FEED:8D 0E C0   548      STA CIRCUITCHAR
FEED:0C FB 04   549      TSB VMODE
FEED:0A         550      PEX
FEED:5A         551      PBY
FEED:20 CD CD   552      JSR CHK80
FEED:7A         553      PLX
FEED:FA         554      *
FEED:A9 FD      555      IOPRT1
FEED:80 C9      556      BRA IOPRT2
FEED:         557      *
FEED:         558      * DECCH decrements the current cursor
FEED:         559      * CLRCCH sets all cursors to 0
FEED:         560      * SETCUR sets cursors to value in Acc.
FEED:         561      * See explanatory note with GETCUR
FEED:         562      *
FEED:5A         563      DECCH
FEED:20 90 CC   564      JSR GETCUR
FEED:88         565      DRY
FEED:80 05      566      BRA SETCUR1
FEED:         567      *
FEED:A9 01      568      CLRCCH
FEED:3A         569      WOTHCCH
FEED:5A         570      SETCUR
FEED:A8         571      TAY
FEED:20 AD CC   572      SETCUR1
FEED:7A         573      PLY
FEED:AD 7B 05   574      LDA OUCRCH
FEF5:60         575      RTS
FEF6:         576      *
FEF6:20 00 FE   577      CROW
FEF9:68         578      PLA
FEFA:68         579      PLA
FEFB:D0 6C      580      BNE MONZ
FEFD:         581      *
;HANDLE CR AS BLANK
; THEN POP STACK
; AND RETURN TO MON
; (ALWAYS)

```

```

22 AUTOST22      Apple //c F8 monitor firmware      20-OCT-86 06:41 PAGE 89
FEF0:60          582 READ      RTS      ;Tape read not needed
FEF0:        583 *          ;
FEF0:        584 * OPTBL is a table containing the new opcodes that
FEF0:        585 * wouldn't fit into the existing lookup table.
FEF0:        586 *
FEF0:12          587 OPTBL    DFB $12      ;ORA (ZPAG)
FEF0:14          588          DFB $14      ;TRB ZPAG
FEF0:1A          589          DFB $1A      ;INC A
FEF0:1C          590          DFB $1C      ;TRB ABS
FEF0:32          591          DFB $32      ;AND (ZPAG)
FEF0:34          592          DFB $34      ;BIT ZPAG,X
FEF0:3A          593          DFB $3A      ;DEC A
FEF0:3C          594          DFB $3C      ;BIT ABS,X
FEF0:52          595          DFB $52      ;EOR (ZPAG)
FEF0:5A          596          DFB $5A      ;PHY
FEF0:64          597          DFB $64      ;STZ ZPAG
FEF0:72          598          DFB $72      ;ADC (ZPAG)
FEF0:74          599          DFB $74      ;STZ ZPAG,X
FEF0:7A          600          DFB $7A      ;PLY
FEF0:7C          601          DFB $7C      ;JMP (ABS,X)
FEF0:88          602          DFB $88      ;BIT IMM
FEF0:92          603          DFB $92      ;STA (ZPAG)
FEF0:9C          604          DFB $9C      ;STZ ABS
FEF0:9E          605          DFB $9E      ;STZ ABS,X
FEF0:A2          606          DFB $A2      ;LDA (ZPAG)
FEF0:A4          607          DFB $A4      ;CMP (ZPAG)
FEF0:A6          608          DFB $A6      ;SBC (ZPAG)
FEF0:A8          609          DFB $A8      ;??? (the unknown opcode)
FEF0:AC          0016 610 NUMOPS EQU *-OPTBL-1 ;number of bytes to check
FEF0:B0          611 *
FEF0:B2          612 * INDX contains pointers to the mnemonics for each of
FEF0:B4          613 * the opcodes in OPTBL. Pointers with BIT 7
FEF0:B6          614 * set indicate extensions to MMEM or MMENR.
FEF0:B8          615 *
FEF0:BA          616 INDX    DFB $38
FEF0:BC          617          DFB $3B
FEF0:BE          618          DFB $37
FEF0:C0          619          DFB $3B
FEF0:C2          620          DFB $39
FEF0:C4          621          DFB $21
FEF0:C6          622          DFB $36
FEF0:C8          623          DFB $21
FEF0:CA          624          DFB $3A
FEF0:CC          625          DFB $F8
FEF0:CE          626          DFB $FA
FEF0:D0          627          DFB $3B
FEF0:D2          628          DFB $FA
FEF0:D4          629          DFB $F9
FEF0:D6          630          DFB $22
FEF0:D8          631          DFB $21
FEF0:DA          632          DFB $3C
FEF0:DC          633          DFB $FA
FEF0:DE          634          DFB $3A
FEF0:E0          635          DFB $3D
FEF0:E2          636          DFB $3E
FEF0:E4          637          DFB $3F
FEF0:E6          638          DFB $3F
FEF0:E8          639          BRK

```

```

22 AUTOST22      Apple //c F8 monitor firmware      20-OCT-86 06:41 PAGE 90
FEF0:60          640 *          LDA $5C5
FEF0:62          641 PRERR    JSR COUNT
FEF0:64          642          LDA #902
FEF0:66          643          JSR COUNT
FEF0:68          644          JSR COUNT
FEF0:6A          645          JSR COUNT
FEF0:6C          646 *          LDA #987
FEF0:6E          647 BELL     JMP COUNT
FEF0:70          648          ;
FEF0:72          649 *          ;
FEF0:74          650 RESTORE  LDA STATUS
FEF0:76          651          PRA
FEF0:78          652          LDA A5H
FEF0:7A          653 RESTR1  LDX YREG
FEF0:7C          654          LDY YREG
FEF0:7E          655          PIP
FEF0:80          656          RTS
FEF0:82          657 *          ;
FEF0:84          658 SAVE     STA A5H
FEF0:86          659 SAV1     STX XREG
FEF0:88          660          STY YREG
FEF0:8A          661          PRP
FEF0:8C          662          PLA
FEF0:8E          663          STA STATUS
FEF0:90          664          TSX
FEF0:92          665          STX SPNT
FEF0:94          666          CLD
FEF0:96          667          RTS
FEF0:98          668 *          ;
FEF0:9A          669 ODRST    JSR SETNORM
FEF0:9C          670          JSR INIT
FEF0:9E          671          JSR SETVID
FEF0:A0          672          JSR SETKBD
FEF0:A2          673 *          ;
FEF0:A4          674 MON      CLD
FEF0:A6          675          JSR BELL
FEF0:A8          676 MONZ    LDA #5AA
FEF0:AA          677          STA PROMPT
FEF0:AC          678          JSR GETLNZ
FEF0:AE          679          JSR ZMODE
FEF0:B0          680 NXITIM   JSR GETNUM
FEF0:B2          681          STY YSAV
FEF0:B4          682          LDY #SUBTEL-CRTEL
FEF0:B6          683 CHSRCH  DEY
FEF0:B8          684          BMI MON
FEF0:BA          685          CMP CRTEL,Y
FEF0:BC          686          BNE CHSRCH
FEF0:BE          687          JSR TOSUB
FEF0:C0          688          LDY YSAV
FEF0:C2          689          JMP NXITIM
FEF0:C4          690 *          ;
FEF0:C6          691 DIG      LDX #903
FEF0:C8          692          ASL A
FEF0:CA          693          ASL A
FEF0:CC          694          ASL A
FEF0:CE          695          ASL A
FEF0:D0          696 NXITIM   ASL A
FEF0:D2          697          ROL AZL

```

22 AUTOST2 Apple //c F8 monitor firmware

```

FF93:26 3F 698 ROL A2H
FF95:CA F8 699 DEX
FF96:10 F8 FF90 BPL NXTBIT
FF98:A5 31 700 LDA MODE
FF9A:00 06 FF92 BNE NXRTRAS
FF9C:B5 3F 702 LDA A2H,X
FF9E:95 3D 704 STA A1H,X
FF9A:95 41 705 STA A3H,X
FFA2:E8 706 NXRTRAS2
FFA3:F0 F3 FF98 NXRTRAS
FFA5:D0 06 FF9A BNE NXRTRCH
FFA7:A2 00 709 GETNUM
FFA8:86 3E 710 LDX #500
FFAB:86 3F 711 STX A2H
FFAD:20 B4 C5 712 NXRTRCH
FFB0:49 B0 713 XOR #580
FFB2:C9 0A 714 CMP #50A
FFB4:90 D4 FF9A BCC DIG
FFB6:69 88 716 ADC #988
FFB8:C9 FA 717 CMP #5FA
FFBA:4C C8 CF 718 JMP LOOKASC
FFBC:00 719 BRK
FFBE: 720 *
FFB8:A9 FE 721 TOSUB
FFC0:48 722 PHA
FFC1:B9 E3 FF FFC1 LDA SUBTEL,Y
FFC3:48 724 PHA
FFC5:A5 31 725 LDA MODE
FFC7:A0 00 726 ZMODE
FFC9:84 31 727 STY MODE
FFCB:60 728 RTS
FFCD: 729 *
FFC8:BC 730 CTRTEL
FFC0:B2 731 DFB
FFC2:82 732 DFB
FFC4:BE 733 DFB
FFC6:9A 734 DFB
FFC8:EF 735 DFB
FFDA:C4 736 DFB
FFDB:88 737 DFB
FFDC:A6 738 DFB
FFDE:A4 739 DFB
FFDF:06 740 DFB
FFE0:95 741 DFB
FFE1:07 742 DFB
FFE2:02 743 DFB
FFE3:05 744 DFB
FFE4:00 745 DFB
FFE5:93 746 DFB
FFE6:A7 747 DFB
FFE7:C6 748 DFB
FFE8:99 749 DFB
FFE9:EC 750 DFB
FFEA:ED 751 DFB
FFEB:EA 752 NOP
FFEC: 753 *
FFED: 754 *
FFEE: 755 *

```

\* Table of low order monitor routine  
 dispatch addresses.

20-OCT-86 06:41 PAGE 91

```

;LEAVE X-STEP IF DIG
;IF MODE IS ZERO,
; THEN COPY A2 TO A1 AND A3
;CLEAR A2
;Get char, iny, upshift
;it's a digit
;+ Check for quote
;DISPATCH TO SUBROUTINE, BY
; PUSHING THE HI-ORDER SUBR ADDR,
; THEN THE LO-ORDER SUBR ADDR,
; ONTO THE STACK,
; (CLEARING THE MODE, SAVE THE OLD
; MODE IN A-REG),
; AND 'RTS' TO THE SUBROUTINE!
;C (BASIC WARM START)
;V (USER VECTOR)
;E (OPEN AND DISPLAY REGISTERS)
;I (Mini assembler)
;V (MEMORY VERIFY)
;K (IN#SLOT)
;P (PRASLOT)
;B (BASIC COLD START)
;S (SUBTRACTION)
;+ (ADDITION)
;M (MEMORY MOVE)
;N (SET NORMAL VIDEO)
;I (SET INVERSE VIDEO)
;L (DISASSEMBLE 20 INSTRS)
;G (EXECUTE PROGRAM)
;F (MEMORY FILL)
;C (ADDRESS DELIMITER)
;Cr (END OF INPUT)
;BLANK
;+S (Step)
;+T (Trace)
;+

```

22 AUTOST2 Apple //c F8 monitor firmware

```

FFB3:B2 756 *
FFB4:C9 757 SUBTEL
FFB5:BE 758 DFB
FFB6:6B 759 DFB
FFB7:35 760 DFB
FFB8:8C 761 DFB
FFB9:96 762 DFB
FFBA:AF 763 DFB
FFBB:17 764 DFB
FFBC:17 765 DFB
FFBD:2B 766 DFB
FFBE:1F 767 DFB
FFBF:83 768 DFB
FFC0:7F 769 DFB
FFC1:5D 770 DFB
FFC2:B5 771 DFB
FFC3:17 772 DFB
FFC4:17 773 DFB
FFC5:F5 774 DFB
FFC6:03 775 DFB
FFC7:70 776 DFB
FFC8:6E 777 DFB
FFC9: 778 *
FFCA: 779 *
FFCB: 780
FFCC: 781 *
FFCD:FB 03 782 DM WMI
FFCE:62 FA 783 DM RESET
FFCF:03 C8 784 DM NEMING
0000: 62 Include bank2

```

20-OCT-86 06:41 PAGE 92

```

;NON-MASKABLE INTERRUPT VECTOR
;RESET VECTOR
;INTERRUPT REQUEST VECTOR

```

```

0000: 2 *****
0000: 3 *
0000: 4 * Bank 2 of the roms
0000: 5 *
0000: 6 *****
0000: ----- NEXT OBJECT FILE NAME IS FIRM.2
C000: 7      org $C000
C000: 63     include mint
C000: 1      ds $C100-*,0

```

```

;Mouse & acia interrupt handler

```

```

C100: 4 *****
C100: 5 *
C100: 6 * Mouse interrupt handler
C100: 7 *
C100: 8 * MOUSEINT - Monitor's interrupt handler
C100: 9 *
C100: 10 * Returns C = 0 if interrupt handled
C100: 11 * If not mouse interrupt, goes to aciaint
C100: 12 * New in this rom:
C100: 13 * If D7 of mousemode = 1, mouse X and Y interrupts are not processed
C100: 14 * and are passed on to the user.
C100: 15 *
C100: 16 *****
C100: 17 mouseint equ *
C100: 18      lda #$0F
C102:1C 7F 07      trb mouseint
19
C105:38      sec
C106:      * Check for vertical blanking interrupt
C106:AD 19 C0      lda vblint
C109:10 2B C136    bpl chkmou
C108:8D 79 C0      sta iouenbl
C10E:A9 0C          lda #vblmode
C110:2C FF 07      bit mousemode
C113:00 03 C118    bne cwmovbl
C115:8D 5A C0      sta iou+2
C118:09 02          ora #mousemode
C11A:8D 78 C0      sta ioudsbl
C110:2C 7F 06      bit mousearm
C120:00 02 C124    bne cwmoved
C122:A9 0C          lda #vblmode
C124:2C 63 C0      bit moubut
C127:10 02 C12B    bpl cwbnt
C129:49 04          eor #butmode
C12B:2D FF 07      cwbnt
C12E:0C 7F 07      tsb mousestat
C131:1C 7F 06      trb mousearm
C134:69 FE          adc #$FE
C136:      * Check & update mouse movement
C136:      43 chkmou equ *
C136:AD FF 07      lda mousemode
C139:30 72 C1AD    bml xadnone
C138:AD 15 C0      lda mouseint
C13E:0D 17 C0      ora mouyint
C141:10 6A C1AD    bpl xadnone
C143:8A          txa
C144:A2 00          ldx #0
C146:2C 15 C0      bit mousexint
C149:30 0A C155    bml cwmov
C14B:98          tya
C14C:49 80          eor #$80
C14E:A2 80          ldx #$80
C150:2C 17 C0      bit mouyint
C153:10 39 C18E    bpl cmov
C155:0A          asl A
C156:BD 7F 04      lda mouse1,x
59      ;A = current low byte

```

```

21      sec
22      * Check for vertical blanking interrupt
23      lda vblint
24      bpl chkmou
25      sta iouenbl
26      lda #vblmode
27      bit mousemode
28      bne cwmovbl
29      sta iou+2
30      ora #mousemode
31      sta ioudsbl
32      bit mousearm
33      bne cwmoved
34      lda #vblmode
35      bit moubut
36      bpl cwbnt
37      eor #butmode
38      cwbnt
39      tsb mousestat
40      trb mousearm
41      adc #$FE
42      * Check & update mouse movement
43      chkmou equ *
44      lda mousemode
45      bml xadnone
46      lda mouseint
47      ora mouyint
48      bpl xadnone
49      txa
50      ldx #0
51      bit mousexint
52      bml cwmov
53      tya
54      eor #$80
55      ldx #$80
56      bit mouyint
57      bpl cmov
58      asl A
59      lda mouse1,x

```

24 MINT	Mouse & serial interrupt stuff	20-OCT-86 06:41 PAGE 95	24 MINT	Mouse & serial interrupt stuff	20-OCT-86 06:41 PAGE 96
60	bcs cmlong		106	* This routine will determine if the source of	
61	cmp m.inhl,x		97	* is either of the built in ACIAs. If neither port	
62	bne cmlong		98	* generated the interrupt, or the interrupt was due	
63	lda mouxh,x		99	* to a transmit buffer empty, protocol converter, or	
64	cmp m.inhl,x		100	* "unbuffered" receiver full, the carry is set indi-	
65	beq cmlong		101	* cating an externally serviced interrupt.	
66	lda mouxl,x		102	* If the interrupt source was keyboard, "buffered"	
67	bne cmlong		103	* serial input, or the DCD, the interrupt is serviced	
68	dec mouxh,x		104	* and the carry is cleared indicating interrupt was	
69	bne cmlong		105	* serviced. (DCD handshake replaces CTS.)	
70	bra cmlong		106	* Location "ACIABUF" specifies which (if either) re-	
71	cmp mouxl,x		107	* ceiver data is buffered. For port 1 it must contain	
72	bne cmlong		108	* SCI, for port 2 a SC2. Any other values are cause	
73	lda mouxh,x		109	* interrupts to pass to external (RAM based) routines.	
74	cmp mouxh,x		110	* Location "TRIPED" specifies whether keyboard in-	
75	beq cmlong		111	* put should be buffered, ignored, or processed by	
76	inc mouxl,x		112	* RAM based routines. If bit 7-1 and bit 6-0, key-	
77	bne cmlong		113	* board data is placed in the type-ahead buffer. If	
78	inc mouxh,x		114	* bit 6 is set the interrupt is cleared, but must	
79	cpb #0		115	* be recognized and serviced by a RAM routine. If	
80	beq cmlong		116	* both bits = 0, the interrupt is serviced, but the	
81	sta mouxl,x		117	* keyboard data is ignored.	
82	lda #mode		118	* While using type-ahead, Open-Apple CTRL-X will	
83	and mode		119	* flush the buffer. No other code is recognized.	
84	beq cmlong		120	* If the source was an ACIA that has the transmit	
85	sta iouenbl		121	* interrupt enabled, the original value of the ACIAs	
86	sta iou3		122	* status registers is preserved. Automatic serial input	
87	sta ioudsbl		123	* buffering is not serviced from a port so configured.	
88	ora #movarm		124	* Interrupts originating from the protocol converter or	
89	tsb mouarm		125	* keyboard (RAM serviced) do not inhibit serial buffering	
90	lda #50E		126	* and are passed thru. The RAM service routine can rec-	
91	and moustat		127	* ognize the interrupt source by a 1 state in bit 6 of	
92	adc #5FE		128	* the ACIAs status register. The RAM service routine must	
93	bcs aciaint		129	* cause the clearing of DSR (bit 6) AND make a second ac-	
94	jmp swtst2		130	* cess to the status register before returning.	
			131		
			132		
			133	notacia sec	;Not acia int
			134	actone rts	
			135	actone equ *	
			136	jsr aciaint2	;Extra jsr since rest needs RTS
			137	jmp swtst2	
			138	actone equ *	
			139	ldx #comslot	;Test port 2 first
			140	jsr aciatst	;Check for interrupt
			141	bcc actone	;Return if interrupt done
			142	dex	;Try port 1
			143	actst	;Get index for acia
			144	lda #54	;If xmit ints enabled pass to user
			145	eor scomd,y	;Check if DCS, DCS = 01
			146	and #50C	
			147	beq notacia	;User better take it!
			148	lda sstat,y	;Get status
			149	sta astat,x	;Save it away
			150	bpl notacia	;No interrupt
			151	cpb #comslot	;C-1 if com port. Called from serout3
			152	bcs alport2	;Invert DSR if port1
			153	eor #540	

```

C1DC:3C 38 05      bit  extint,x
C1DE:70 29 C20A 155 hvs  aipass
C1E1:10 25 C208 156 bpl  alaitit
C1E3:90 23 C208 157 bcc  alaitit
C1E5:89 40 158     bit  $540
C1E7:F0 21 C20A 159 beq  aipass
C1E9:      160 * It's a keyboard interrupt
C1E9:AD 00 C0 161 lda  rbd
C1EC:A0 80 162 ldy  $580
C1EE:20 28 C2 163 jsr  putbuf
C1F1:C9 96 164 cmp  $598
C1F3:D0 08 C200 165 bne  ainoflsh
C1F5:AD 62 C0 166 lda  butnl
C1F8:10 06 C200 167 bpl  ainoflsh
C1FA:8E FC 05 168 stx  twkey
C1FD:8E FF 06 169 stx  trkey
C200:AD 10 C0 170 ainoflsh lda  kbdsrb
C203:      171 * $A0-$50 cable needed by serial firmware
C203:      C142 172 devno2 equ  *-sldmty
C203:AD 00 B0 173 ldy  $580
C205:B9 F9 BF 174 lda  sstat,y
C208:29 BF 175 aleaitit and  $5BF
C20A:0A 176 aipass  asl  A
C20B:0A 177     asi  A
C20C:29 20 178 and  $520
C20E:F0 3E C24E 179 beq  aciadone
C210:10 B9 FA BF 180 lda  scomd,y
C213:49 01 181 eor  #1
C215:29 03 182 and  #3
C219:8A 184     txa
C21A:4D FC 04 185 eor  aciabuf
C21D:D0 93 C1B2 186 bne  notacia
C21F:08 187     php
C220:20 22 C3 188 jsr  getdata
C223:90 28 C24D 189 bcc  aleat
C225:A0 00 190 ldy  #0
C227:D0 191 dfb  $D0
C228:      192 putbuf equ  *
C228:08 193 php
C229:DA 194 phx
C22A:48 195 pha
C22B:B9 7C 05 196 lda  twser,y
C22E:AA 197     tax
C22F:1A 198 inc  A
C230:89 7F 199 bit  $57F
C232:D0 01 C235 bne  pbok
C234:98 201 tya
C235:D9 7C 06 202 pbok cmp  trser,y
C238:F0 03 C23D 203 beq  pbfull
C23A:99 7C 05 204 sta  twser,y
C23D:68 205 pbfull pla
C23E:2C 14 C0 206 bit  rdramwrt
C241:8D 05 C0 207 sta  wradram
C244:9D 00 08 208 sta  tibuf,x
C247:30 03 C24C 209 bml  alaux
C249:8D 04 C0 210 sta  wrmainram
C24C:FA 211 alaux  plx

```

;Get DSR status back

212 aleat plp

213 aciadone rts

```

Mouse & serial interrupt stuff      20-OCT-96   06:41 PAGE 99

215 *****
216 * C2AF:
217 * C2AF:
218 * C2AF:
219 * C2AF:
220 *****
221 serout3 equ *
222 C2AF:20 55 C2 serout4
223 C2AF:4C 84 C7 jmp swt32
224 serout4 equ *
225 C255:48 pha
226 C256:2C AB C2 bit
227 C259:F0 03 C25E sbrd
228 C25B:FE 38 07 inc col,x
229 sordy jsr getstat2
230 C25E:20 B2 C2 and #530
231 C263:C9 10 cmp #510
232 C265:D0 F7 bne sordy
233 C267:BD B8 06 lda flags,x
234 C26A:89 20 bit #520
235 C26C:F0 1F C2BD beq sook
236 C26E:EC FC 04 cpx aciabuf
237 C271:F0 13 C2B6 beq sotst
238 C273:20 E9 C2 jsr xrdnbuf
239 C276:90 0E C2B6 bcc sotst
240 C278:BC 34 C2 ldy charptr,x
241 C27B:99 FE 05 ldy sta charbuf,y
242 C27E:BD B8 06 lda flags,x
243 C281:09 04 ora #504
244 C283:9D B8 06 sta flags,x
245 C286:BD B8 06 lda flags,x
246 C289:29 02 and #502
247 C28B:D0 D1 bne sordy
248 C290:BC 42 C1 ldy devo2,x
249 C290:68 pla
250 C292:91:48 pha
251 C292:99:F8 BF sta sdata,y
252 C295:3C B8 06 bit flags,x
253 C298:49 0D eor #500
254 C29A:0A 00 asl A
255 C29B:9D 0D C2A8 bne sodone
256 C29D:50 16 C2A5 bvc clrcol
257 C29F:A9 14 C2A7 lda #514
258 C2A1:6A ror A
259 C2A2:20 55 C2 jsr serout4
260 C2A5:64 24 C2A6 clrcol
261 C2A7:9E 38 07 str ch
262 C2AA:68 pla
263 C2AB:60 sorts
264 C2AC:
265 C2AC:
266 * C2AC:
267 * C2AC:
268 * C2AC:
269 * C2AC:
270 * C2AC:
271 *****
272 * C2AC:
273 * C2AC:
274 * C2AC:
275 * C2AC:
276 *****
277 * C2AC:
278 * C2AC:
279 * C2AC:
280 * C2AC:
281 * C2AC:
282 *****
283 * C2AC:
284 * C2AC:
285 * C2AC:
286 * C2AC:
287 * C2AC:
288 *****
289 * C2AC:
290 * C2AC:
291 * C2AC:
292 * C2AC:
293 * C2AC:
294 *****
295 * C2AC:
296 * C2AC:
297 * C2AC:
298 * C2AC:
299 * C2AC:
300 *****
301 * C2AC:
302 * C2AC:
303 * C2AC:
304 * C2AC:
305 * C2AC:
306 *****
307 * C2AC:
308 * C2AC:
309 * C2AC:
310 * C2AC:
311 * C2AC:
312 *****
313 * C2AC:
314 * C2AC:
315 * C2AC:
316 * C2AC:
317 * C2AC:
318 *****
319 * C2AC:
320 * C2AC:
321 * C2AC:
322 * C2AC:
323 *****
324 * C2AC:
325 * C2AC:
326 * C2AC:
327 * C2AC:
328 * C2AC:
329 *****
330 * C2AC:
331 * C2AC:
332 * C2AC:
333 * C2AC:
334 * C2AC:
335 *****
336 * C2AC:
337 * C2AC:
338 * C2AC:
339 * C2AC:
340 * C2AC:
341 *****
342 * C2AC:
343 * C2AC:
344 * C2AC:
345 * C2AC:
346 * C2AC:
347 *****
348 * C2AC:
349 * C2AC:
350 * C2AC:
351 * C2AC:
352 * C2AC:
353 *****
354 * C2AC:
355 * C2AC:
356 * C2AC:
357 * C2AC:
358 * C2AC:
359 *****
360 * C2AC:
361 * C2AC:
362 * C2AC:
363 * C2AC:
364 * C2AC:
365 *****
366 * C2AC:
367 * C2AC:
368 * C2AC:
369 * C2AC:
370 * C2AC:
371 *****
372 * C2AC:
373 * C2AC:
374 * C2AC:
375 * C2AC:
376 * C2AC:
377 *****
378 * C2AC:
379 * C2AC:
380 * C2AC:
381 * C2AC:
382 * C2AC:
383 *****
384 * C2AC:
385 * C2AC:
386 * C2AC:
387 * C2AC:
388 * C2AC:
389 *****
390 * C2AC:
391 * C2AC:
392 * C2AC:
393 * C2AC:
394 * C2AC:
395 *****
396 * C2AC:
397 * C2AC:
398 * C2AC:
399 * C2AC:
400 * C2AC:
401 *****
402 * C2AC:
403 * C2AC:
404 * C2AC:
405 * C2AC:
406 * C2AC:
407 *****
408 * C2AC:
409 * C2AC:
410 * C2AC:
411 * C2AC:
412 * C2AC:
413 *****
414 * C2AC:
415 * C2AC:
416 * C2AC:
417 * C2AC:
418 * C2AC:
419 *****
420 * C2AC:
421 * C2AC:
422 * C2AC:
423 * C2AC:
424 * C2AC:
425 *****
426 * C2AC:
427 * C2AC:
428 * C2AC:
429 * C2AC:
430 * C2AC:
431 *****
432 * C2AC:
433 * C2AC:
434 * C2AC:
435 * C2AC:
436 * C2AC:
437 *****
438 * C2AC:
439 * C2AC:
440 * C2AC:
441 * C2AC:
442 * C2AC:
443 *****
444 * C2AC:
445 * C2AC:
446 * C2AC:
447 * C2AC:
448 * C2AC:
449 *****
450 * C2AC:
451 * C2AC:
452 * C2AC:
453 * C2AC:
454 * C2AC:
455 *****
456 * C2AC:
457 * C2AC:
458 * C2AC:
459 * C2AC:
460 * C2AC:
461 *****
462 * C2AC:
463 * C2AC:
464 * C2AC:
465 * C2AC:
466 * C2AC:
467 *****
468 * C2AC:
469 * C2AC:
470 * C2AC:
471 * C2AC:
472 * C2AC:
473 *****
474 * C2AC:
475 * C2AC:
476 * C2AC:
477 * C2AC:
478 * C2AC:
479 *****
480 * C2AC:
481 * C2AC:
482 * C2AC:
483 * C2AC:
484 * C2AC:
485 *****
486 * C2AC:
487 * C2AC:
488 * C2AC:
489 * C2AC:
490 * C2AC:
491 *****
492 * C2AC:
493 * C2AC:
494 * C2AC:
495 * C2AC:
496 * C2AC:
497 *****
498 * C2AC:
499 * C2AC:
500 * C2AC:
501 * C2AC:
502 * C2AC:
503 *****
504 * C2AC:
505 * C2AC:
506 * C2AC:
507 * C2AC:
508 * C2AC:
509 *****
510 * C2AC:
511 * C2AC:
512 * C2AC:
513 * C2AC:
514 * C2AC:
515 *****
516 * C2AC:
517 * C2AC:
518 * C2AC:
519 * C2AC:
520 * C2AC:
521 *****
522 * C2AC:
523 * C2AC:
524 * C2AC:
525 * C2AC:
526 * C2AC:
527 *****
528 * C2AC:
529 * C2AC:
530 * C2AC:
531 * C2AC:
532 * C2AC:
533 *****
534 * C2AC:
535 * C2AC:
536 * C2AC:
537 * C2AC:
538 * C2AC:
539 *****
540 * C2AC:
541 * C2AC:
542 * C2AC:
543 * C2AC:
544 * C2AC:
545 *****
546 * C2AC:
547 * C2AC:
548 * C2AC:
549 * C2AC:
550 * C2AC:
551 *****
552 * C2AC:
553 * C2AC:
554 * C2AC:
555 * C2AC:
556 * C2AC:
557 *****
558 * C2AC:
559 * C2AC:
560 * C2AC:
561 * C2AC:
562 * C2AC:
563 *****
564 * C2AC:
565 * C2AC:
566 * C2AC:
567 * C2AC:
568 * C2AC:
569 *****
570 * C2AC:
571 * C2AC:
572 * C2AC:
573 * C2AC:
574 * C2AC:
575 *****
576 * C2AC:
577 * C2AC:
578 * C2AC:
579 * C2AC:
580 * C2AC:
581 *****
582 * C2AC:
583 * C2AC:
584 * C2AC:
585 * C2AC:
586 * C2AC:
587 *****
588 * C2AC:
589 * C2AC:
590 * C2AC:
591 * C2AC:
592 * C2AC:
593 *****
594 * C2AC:
595 * C2AC:
596 * C2AC:
597 * C2AC:
598 * C2AC:
599 *****
600 * C2AC:
601 * C2AC:
602 * C2AC:
603 * C2AC:
604 * C2AC:
605 *****
606 * C2AC:
607 * C2AC:
608 * C2AC:
609 * C2AC:
610 * C2AC:
611 *****
612 * C2AC:
613 * C2AC:
614 * C2AC:
615 * C2AC:
616 * C2AC:
617 *****
618 * C2AC:
619 * C2AC:
620 * C2AC:
621 * C2AC:
622 * C2AC:
623 *****
624 * C2AC:
625 * C2AC:
626 * C2AC:
627 * C2AC:
628 * C2AC:
629 *****
630 * C2AC:
631 * C2AC:
632 * C2AC:
633 * C2AC:
634 * C2AC:
635 *****
636 * C2AC:
637 * C2AC:
638 * C2AC:
639 * C2AC:
640 * C2AC:
641 *****
642 * C2AC:
643 * C2AC:
644 * C2AC:
645 * C2AC:
646 * C2AC:
647 *****
648 * C2AC:
649 * C2AC:
650 * C2AC:
651 * C2AC:
652 * C2AC:
653 *****
654 * C2
```

C2C3:  
C2C3:  
C2C3:  
C2C3:  
C2C3:  
C2C3:  
C2C3:  
C2C3:  
C2C3:20 C9 C2  
C2C6:4C 84 C7  
C2C9:  
C2C9:FC FC 0A

C2D0:20 FD C2  
C2D3:B0 1F C  
C2D5:  
C2D5:BD B8 06  
C2D8:89 04  
C2DA:F0 0D C  
C2DC:28 8B

CZDE:9D B8 06  
C2E1:BC 34 C2  
C2FA:B8 FF 05

CZE1:38  
CZE8:60  
CZE9:

CZE9:20 BZ CZ  
C2EC:29 08  
C2EF:18

C2F1:20 22 C3  
C2F4:60

C2F5: 00 80  
C2F5:00 80

C2F7:  
C2F7:  
C2F7:

:LF72  
 :LF72  
 :LF72

C2F7: 20 FD C2

C2FD: C  
C2FD:B9 7C 06

C303:18  
C304:F0 1B C

C307:1A  
C308:89 7F

030C:98

C34E:	4	*****	5	NAME	: MOVEAUX	*****
C34E:	6	*****	6	FUNCTION:	PERFORM CROSSBANK MEMORY MOVE	*****
C34E:	7	*****	7	INPUT	: AI-SOURCE ADDRESS	*****
C34E:	8	*****	8	*****	A2-SOURCE END	*****
C34E:	9	*****	9	*****	AI-DESTINATION START	*****
C34E:	10	*****	10	*****	CARRY SET-MAIN->CARD	*****
C34E:	11	*****	11	*****	CUR-CARD->MAIN	*****
C34E:	12	*****	12	OUTPUT	: NONE	*****
C34E:	13	*****	13	VOLATILE:	NOTHING	*****
C34E:	14	*****	14	CALLS	: NOTHING	*****
C34E:	15	*****	15	*****	*****	*****
C34E:	16	*****	16	MOVEAUX	ECU *	*****
C34E:	17	*****	17	PHA	PHA	*****
C34E:	18	*****	18	PHA	RDRAWM	*****
C34E:	19	*****	19	PHA	RDRAWM	*****
C34E:	20	*****	20	LOA	RDRAWMRT	*****
C34E:	21	*****	21	PHA	PHA	*****
C34E:	22	*****	22	*****	*****	*****
C34E:	23	*****	23	*****	*****	*****
C34E:	24	*****	24	*****	*****	*****
C34E:	25	*****	25	BCC	MOVEC2M ;->CARD->MAIN	*****
C34E:	26	*****	26	STA	RDRAWMRT ; SET FOR MAIN	*****
C34E:	27	*****	27	STA	RDRAWMRT ; TO CARD	*****
C34E:	28	*****	28	BCC	MOVESTRT ;-> (ALWAYS TAKEN)	*****
C34E:	29	*****	29	*****	*****	*****
C34E:	30	*****	30	MOVEC2M	ECU *	*****
C34E:	31	*****	31	STA	RDRAWMRT ; SET FOR CARD	*****
C34E:	32	*****	32	STA	RDRAWMRT ; TO MAIN	*****
C34E:	33	*****	33	*****	*****	*****
C34E:	34	*****	34	MOVESTRT	ECU *	*****
C34E:	35	*****	35	MOVELOOP	LOA (A1L) ;get a byte	*****
C34E:	36	*****	36	STA	RDRAWMRT ;move it	*****
C34E:	37	*****	37	INC	A1L	*****
C34E:	38	*****	38	BNE	NEXTAL	*****
C34E:	39	*****	39	INC	A1H	*****
C34E:	40	*****	40	NEXTAL	LOA A1L	*****
C34E:	41	*****	41	CMF	A2L	*****
C34E:	42	*****	42	LOA	A1H	*****
C34E:	43	*****	43	SBC	A2H	*****
C34E:	44	*****	44	INC	A1L	*****
C34E:	45	*****	45	BNE	CO1	*****
C34E:	46	*****	46	INC	A1H	*****
C34E:	47	*****	47	CO1	MOVELOOP	*****
C34E:	48	*****	48	*****	*****	*****
C34E:	49	*****	49	STA	RDRAWMRT	*****
C34E:	50	*****	50	PLA	CO3	*****
C34E:	51	*****	51	BPL	CO3	*****
C34E:	52	*****	52	STA	RDRAWMRT	*****
C34E:	53	*****	53	CO3	ECU *	*****
C34E:	54	*****	54	STA	RDRAWMRT	*****
C34E:	55	*****	55	PLA	RDRAWMRT	*****
C34E:	56	*****	56	BPL	MOVESTRT	*****
C34E:	57	*****	57	STA	RDRAWMRT	*****
C34E:	58	*****	58	MOVESTRT	ECU *	*****
C34E:	59	*****	59	CO3	RDRAWMRT	*****
C34E:	60	*****	60	JMP	RDRAWMRT	*****

```

C3C6: 3 *****
C3C6: 4 * Here is the rest of the diagnostic stuff
C3C6: 5 * the first part has been moved into the $D000 space
C3C6: 6 * to make desperately needed room
C3C6: 7 *
C3C6: 8 *
C3C6: 9 *****
C3C6: 10 TSTMEM equ *
C3C6: 11 stx $01
C3C6: 12 stx $02
C3C6: 13 stx $03
C3C6: 14 ldx #4
C3C6: 15 stx $04
C3C6: 16 MEM1 STA $05
C3C6: 17 ldx #4
C3C6: 18 stx $01
C3C6: 19 inc 1
C3C6: 20 mem2 tay
C3C6: 21 sta lbank2
C3C6: 22 sta lbank2
C3C6: 23 ldx $01
C3C6: 24 and #$F0
C3C6: 25 cmp #$C0
C3C6: 26 bne mem3
C3C6: 27 ldx lbank1
C3C6: 28 ldx lbank1
C3C6: 29 ldx $01
C3C6: 30 adc #F
C3C6: 31 bne mem4
C3C6: 32 mem3 ldx $01
C3C6: 33 mem4 sta $03
C3C6: 34 tya
C3C6: 35 ldy #500
C3C6: 36 mem5 clc
C3C6: 37 adc
C3C6: 38 sta ($02),Y
C3C6: 39 dex
C3C6: 40 bpl mem6
C3C6: 41 ldx #4
C3C6: 42 mem6
C3C6: 43 bne mem5
C3C6: 44 inc 1
C3C6: 45 bne mem2
C3C6: 47 inc $01
C3C6: 48 ldx #4
C3C6: 49 ldx $05
C3C6: 50 mem7 tay
C3C6: 51 ldx lbank2
C3C6: 52 ldx lbank2
C3C6: 53 ldx $01
C3C6: 54 and #$F0
C3C6: 55 cmp #$C0
C3C6: 56 bne mem8
C3C6: 57 ldx lbank1
C3C6: 58 ldx $01
C3C6: 59 adc #F
C3C6: 60 bne mem9

```

```

C42A: A5 01 61 mem8 ldx $01
C42C: A5 03 62 mem9 sta $03
C42E: 98 63 tya
C42F: A0 00 64 ldy #500
C431: 18 65 memA clc
C432: 7D 2A C8 66 adc ntbl,x
C433: 51 02 67 eor ($02),Y
C437: D0 39 C472 68 bne MEMERROR
C439: B1 02 69 ldx ($02),Y
C43B: CA 70 dex
C43C: 10 02 C440 71 bpl mem8
C43E: A2 04 72 ldx #4
C440: C8 73 memB iny
C441: D0 EE C431 74 bne memA
C443: D6 01 75 inc 1
C445: D0 C8 C412 76 bne mem7
C447: 6A 77 ror 3
C448: 2C 19 C0 78 bit rdhblbar
C449: 10 02 C44F 79 bpl memC
C44D: A5 80 eor #5A5
C44F: C6 04 81 memC dec $04
C451: 30 03 C456 82 bmi memD
C453: D0 C3 83 jmp mem1
C456: AA 85 memD TAX
C457: 2C 13 C0 86 BIT rdcard
C458: 30 10 C46C 87 BMI MEMF
C45C: 8A 88 tra
C45D: 8D 05 C0 89 STA wcardram
C460: 8D 03 C0 90 STA rdcardram
C463: 8D 09 C0 91 STA setaltzp
C466: 8D 81 C0 92 STA ROMEN
C469: 4C 97 D4 93 jmp TSTZNG
C46C: 8D 08 C0 95 MEMF STA setstdzp
C46F: 4C EF C4 96 JMP SWCFIST

```

[illegible]

26	BWNC22	Apple //c	diagnostic
C4CA:2A	02	143	BADSWTCH
C4CC:7A		144	ply
C4CD:08		145	phb
C4CE:3D	6C	146	bwstchl
C4CF:28		147	plp
C4D0:08		148	
C4D3:9D	03	149	bcd
C4D5:3D	6F	150	bcd
C4D8:0C	06	151	bwstch2
C4DA:9D	08	152	cpy
C4DC:0C	08	153	cpy
C4DE:9D	04	154	bcd
C4E0:0C	11	155	cpy
C4E2:9D	03	156	bcd
C4E4:BD	72	157	bwstch2a
C4E7:9D	88	158	bwstch3
C4EA:CA		159	dex
C4EB:10	E1	160	bpl
C4ED:3D	FE	161	hwy
C4F2:AD	01	163	SWCHST
C4F3:4A	7F	164	swstc1
C4F3:6A		165	swstc2
C4F4:BE	2F	166	lcr
C4F7:FD	0F	167	beq
C4F9:9D	03	168	bcd
C4FB:BE	41	169	bcd
C4FE:9D	FF	170	swstc3
C501:CB		171	bne
C502:00	EF	172	iny
C504:		173	*
C504:AE	30	174	click
C507:2A		175	rol
C508:88		176	swstc4
C509:BE	53	177	beq
C50C:F0	13	178	bcd
C50E:3D	F4	179	bcd
C510:2A		180	rol
C511:9D	07	181	bcd
C513:1E	00	182	asl
C516:9D	EE	183	bcd
C518:9D	EE	184	bcd
C51A:1E	00	185	swstc5
C51D:9D	18	186	bcd
C51F:9D	E7	187	bcd
C521:		188	*
C521:2A		189	rol
C522:CB		190	iny
C523:38		191	sec
C524:BE	01	192	swstc
C526:8D	C8	193	bcd
C528:88		194	dec
C529:F0	08	195	beq
C52B:CB	08	196	cpy
C52D:00	10	197	bne
C52F:AD	B1	198	lcr
C531:00	BE	199	bne

```

20-OCT-86 06:41 PAGE 108

anticipate MMU error

    branch if not IOU error
anticipate IOU error
    compare with where we left off
    skip if MMU

    skip if CPU (loundis or dhires failure)

    skip if IOU
    CPU error (loundis failure)

    print "MMU", "IOU" or "CPU"
    branch forever

    set IOU/MMU switches to match A

    branch if done setting switches
    branch if setting switch to 0-state
    else get index to set switch to 1
    set switch

    branch always taken...

now verify the settings just made
branch if done this pass
branch if this switch no to be verified..

branch always

branch always

restore original value
; and IOU/MMU index

entry next pattern

was MMU just tested?
yes, go test IOU
was IOU just tested?
no, go loop again
no, go test IOUDIS switch
branch always

```

```

C533:A0 09      200 swst7      ldy #IOUIDX
C535:D0 BA      201      bne swst1
C537:          202 *          phy
C537:5A        203 swerr      phy
C538:          204 ;
C538:A2 00      205      ldx #0
C53A:C0 0A      206      cpy #IOUIDX+1
C53C:4C 7D C4   207      jmp bbits1

```

```

;branch always
;save y to distinguish from MMU or GPU
failure
;indicate switch error
;set carry if IOU was cause

```

```

;clear screen for success message

```

```

;test for both Open and Closed Apple
; pressed
;put result in carry

```

```

;put success message on the screen
;Loop forever

```

```

C53F:46 80      209 BIGLOOP  lsr $80
C541:D0 AC      210      bne SWCHTST
C543:A9 A0      211 blp2     lda $A0
C545:A0 00      212      ldy #0
C547:99 00 04   213 blp3     sta $400,y
C54A:99 00 05   214      sta $500,y
C54D:99 00 06   215      sta $600,y
C550:99 00 07   216      sta $700,y
C553:C8         217      iny
C554:D0 F1 C547 218      bne blp3
C556:AD 61 C0   219 blp4     lda butn0
C559:2D 62 C0   220      AND butn1
C55C:0A         221      asl a
C55D:56 FF      222      JNC $FF
C55F:A5 FF      223      LDA $FF
C561:90 03 C566 224      bcc dquit
C563:4C 8E D4   225      jmp DIAGS
C566:          226 *
C566:AD 51 C0   227 dquit    lda txtset
C569:A0 08      228      ldy #8
C56B:59 75 C8   229 suc2     lda success,y
C56E:99 B8 05   230      sta SCREEN,y
C571:88         231      dey
C572:10 F7 C568 232      bpl suc2
C574:30 E0 C556 233      bmi blp4
C576:          235      ds $C580-*, $00
C580:          66      include rw.slinky

```

27 RW,SLINKY	Apple //c diagnostics	20-OCT-86 06:41 PAGE 111	Apple //c diagnostics	60 prbad	lda #badblk
C580:	2 *****			61	error
C580:	3 * PRBAD - Reads bytes from card into the Apple			62 prbadz	sta romin
C580:	4 * D7 of the address = 1 if aux ram			63	rts
C580:	5 *****				
	7 sl,preadd phx	:save x			
C580:DA	8 ldx C581:AE 78 06	:get language card state			
C581:AE	9 inc \$C000,x	:restore it, the rom is anyway			
C581:FE	10 plx	:restore x			
C581:FA	12 lda paddr	:Move the address			
C588:A5	13 sta addri,x				
C58A:9D	14 lda paddr+1				
C58D:A5	15 sta addri,x				
C58F:9D	16 lda paddr+2				
C592:A5	17 and #7F	:Mask off high bit			
C594:29	18 cmp numbanks,y	:Valid address			
C596:D9	19 bge prbad				
C599:B0	20 sta addri,x	:Save current bank			
C59B:FA	21 bit dramwrt				
C59F:2C	22 bit				
C5A1:08	23 php	:Assume main			
C5A2:8D	24 sta wmainram	:If D7 = 1 then aux			
C5A5:24	25 bit paddr+2				
C5A7:10	26 bpl pmain	:Its the card ram			
C5A9:05	27 sta wrdram				
C5AB:8D	28 ldy #6	:More than a page to move?			
C5AD:A5	29 sta yval				
C5B0:8D	30 beq priast	:Get a byte			
C5B3:F0	31 prioop				
C5B5:8D	32 lda data,x				
C5B8:91	33 sta (pbuff),y				
C5BA:C8	34 iny				
C5BB:8D	35 lda data,x				
C5BE:91	36 sta (pbuff),y				
C5C0:C8	37 iny				
C5C1:D0	38 bne prioop	:Bump buffer pointer to next page			
C5C3:56	39 inc pbuff+1	:Dec page count			
C5C5:C6	40 dec pcount+1				
C5C7:D9	41 bne prioop	:Any bytes left to do?			
C5C9:A5	42 lda pcount				
C5CB:F0	43 beq prdone	:Save bytes moved			
C5CD:8D	44 sta xval	:C = 1 if odd # of bytes			
C5D0:4A	45 lsr A				
C5D1:80	46 bcs prodd				
C5D3:8D	47 lda data,x				
C5D6:91	48 sta (pbuff),y				
C5D8:C8	49 iny				
C5D9:8D	50 lda data,x				
C5DC:91	51 sta (pbuff),y				
C5DE:C8	52 iny				
C5DF:CA	53 cpy pcount				
C5E1:D0	54 bne prioop2				
C5E3:8D	55 sta wmainram	:Fix main / aux ram			
C5E6:28	56 plp				
C5E7:10	57 bpl pmain2				
C5E9:8D	58 sta wrdram				
C5EC:80	59 bra prbadz				



27 SW.SLINKY	Apple //c diagnostics	20-OCT-86 06:41 PAGE 115	28 MOODE.X.AUX	Apple //c diagnostics	20-OCT-86 06:41 PAGE 116
C684:DA	178 swsl.bt phx	;save x	C71C:	2 *****	
C685:20 16 C8	179 jsr getlc	;get language state	C71C:	3 * the following code had better start at \$C71C or else	
C686:5A	180 phy	;save it	C71C:	4 *****	
C689:8C 78 06	181 sty sl.lcstate	;save it here too			
C68C:20 EF D8	182 jsr boot.sl	;do the boot			
C68E:4C 0E C8	183 jmp fixlc	;restore language card state and return			
C6C2:DA	185 sw.setaou phx	;save x	C71F:8D 28 C0	6 sta rombank	
C6C3:20 16 C8	186 jsr getlc	;get language card state	C71F:4C C2 C6	7 jmp sw.setaou	;do the real thing
C6C6:5A	187 phy	;save it			
C6C7:20 21 D6	188 jsr x.setaou	;set the mouse mode to a	C722:8D 28 C0	9 sta rombank	
C6CA:4C 0E C8	189 jmp fixlc	;restore language card state and return	C725:4C CD C6	10 jmp sw.slstint	;do the real thing
C6CD:DA	191 sw.slstint phx	;save x			
C6C3:20 16 C8	192 jsr getlc	;get language card state	C728:8D 28 C0	12 sta rombank	
C6D1:5A	193 phy	;save it	C72B:4C D8 C6	13 jmp sw.aread	;do the real thing
C6D2:20 C2 D6	194 jsr x.slstint	;check mouse status bits			
C6D5:4C 0E C8	195 jmp fixlc	;restore language card state and return	C72E:8D 28 C0	15 sta rombank	
C6D8:DA	197 sw.mread	;save x	C731:4C E3 C6	16 jmp sw.mclear	;do the real thing
C6D9:20 16 C8	198 jsr getlc	;get language card state			
C6DC:5A	199 phy	;save it	C734:8D 28 C0	18 sta rombank	
C6D0:20 79 D6	200 jsr x.mread	;updates the mouse screen holes	C737:4C E2 C6	19 jmp sw.mclamp	;do the real thing
C6E0:4C 0E C8	201 jmp fixlc	;restore language card state and return			
C6E3:DA	203 sw.mclear phx	;save x	C73A:8D 28 C0	21 sta rombank	
C6E4:20 16 C8	204 jsr getlc	;get language card state	C73D:4C F9 C6	22 jmp sw.mhome	;do the real thing
C6E7:5A	205 phy	;save it			
C6E8:20 68 D6	206 jsr x.mclear	;sets the mouse to 0,0	C740:8D 28 C0	24 sta rombank	
C6EB:4C 0E C8	207 jmp fixlc	;restore language card state and return	C743:4C 04 C7	25 jmp sw.initmouse	;do the real thing
C6EE:DA	209 sw.mclamp phx	;save x			
C6EF:20 16 C8	210 jsr getlc	;get language card state	C746:8D 28 C0	27 m.oveirq sta rombank	
C6F2:5A	211 phy	;save it	C749:4C 9A C7	28 jmp m.oveirq	
C6F3:20 A3 D6	212 jsr x.mclamp	;store new mouse bounds			
C6F6:4C 0E C8	213 jmp fixlc	;restore language card state and return	C74C:8D 28 C0	30 sta rombank	
C6F9:DA	215 sw.mhome phx	;save x	C74F:4C B4 C6	31 jmp swsl.bt	
C6FA:20 16 C8	216 jsr getlc	;get language card state			
C6FD:5A	217 phy	;save it	C752:8D 28 C0	33 sta rombank	
C6FE:20 51 D6	218 jsr x.mhome	;clear mouse position and status	C755:DA	34 phx	;save x
C701:4C 0E C8	219 jmp fixlc	;restore language card state and return	C756:20 16 C8	35 jsr	;get language card state
C704:DA	221 sw.initmouse phx	;save x	C759:5A	36 phy	;save it
C705:20 16 C8	222 jsr getlc	;get language card state	C75A:8C 78 06	37 sty sl.lcstate	;save it here too
C708:5A	223 phy	;save it	C75D:20 00 D8	38 jsr execute	;do something with slinky
C709:20 00 D6	224 jsr initmouse	;reset the mouse	C760:4C 0E C8	39 jmp fixlc	;restore language card state and return
C70C:4C 0E C8	225 jmp fixlc	;restore language card state and return			
C70F:	000D 227 ds \$C71C-*,00		C763:	41 ds \$C780-*, \$00	
C71C:	67 include moode.x.aux		C780:	68 include switcher2	;Bank switch stuff @ 2:C780

```

C780: 0000      2      ds      $C780-4,$00
C780:          3      *
C780:          4      *
C780:          5      * SWITCHING ROUTINES
C780:          6      *
C780:          7      *
C780:          8      swrl2 sta rombank
C780:          9      swrl2 rti
C780:         10      swrl2 sta rombank
C780:         11      swrl2 rts
C780:         12      swrl2 sta rombank
C780:         13      swrl2 jmp reset
C780:         14      swrl2 sta rombank
C780:         15      swrl2 bit swrlscop
C780:         16      swrl2 jmp irqent
C780:         17      swrl2 sta rombank
C780:         18      swrl2 jmp pcov
C780:         19      swrl2 sta rombank
C780:         20      swrl2 jmp basicin
C780:         21      swrl2 sta rombank
C780:         22      swrl2 jmp swrlsc3
C780:         23      swrl2 sta rombank
C780:         24      swrl2 jmp swrlsc3
C780:         25      swrl2 sta rombank
C780:         26      swrl2 jmp movaux
C780:         27      swrl2 sta rombank
C780:         28      swrl2 jmp xfer
C780:         29      swrl2 sta rombank
C780:         30      swrl2 jmp mouseint
C780:         31      swrl2 sta rombank
C780:         32      swrl2 jmp diags
C780:         33      swrl2 sta rombank
C780:         34      swrl2 jmp atalk
C780:         35      swrl2 sta rombank
C780:         36      swrl2 jmp serout3
C780:         37      swrl2 sta rombank
C780:         38      swrl2 jmp getstat
C780:         39      swrl2 sta rombank
C780:         40      swrl2 jmp xrdser
C780:         41      swrl2 sta rombank
C780:         42      swrl2 jmp getbuf
C780:         43      swrl2 sta rombank
C780:         44      swrl2 jmp zzm
C780:         45      swrl2 jmp swrlsc2
C780:         46      swrl2 jmp ($3DD)
C780:         47      swrl2 jmp getic
C780:         48      swrl2 jmp
C780:         49      swrl2 ply
C780:         50      swrl2 setterm
C780:         51      swrl2 bra fixic
C780:         52      swrl2 ds      $C803-4,$0
C780:         53      swrl2 jmp swrl2
C780:         54      swrl2
C806:DA      56      swrl2 phx
C807:20      57      jmp getic
C80A:5A      58      jmp ply
C80B:20      59      jmp command

```

Command processor for serial &amp; comm 20-OCT-86 06:41 PAGE 120

60	cmp	#uspace	:is it a space? (uppercased)
61	bne	incmd3	:no, go on with 2-chr cmd and handling
62	clic		:yes, ignore spaces between characters
63	:		of 2-chr commands
64	p1a	nowcmd2	:pull uppercased char off stack
65	bra	nowcmd2	:ie mark them "handled" and don't
66	:		do anything else
68	lda	sermode,x	:get sermode back
69	p1a		:save sermode for a minit
70	and	#7	:throw out all but bits 0-2
71	temp		:save - this is index of which cmd it is
72	sta	temp	:get sermode back
73	lra	#5F0	:now clear bits 0-3
74	sta	sermode,x	:since we're done with them now
75	p1a		:get character back
76	phx		:shove x (Cn) on stack
77	ldx	temp	:get index to command's 1st chr
78	cmp	#545	:is it an E?
79	beq	enable	:yes
80	cmp	#544	:no, is it a D?
81	beq	disable	:yes
82	plx		:retrieve X-Cn (old X still in temp)

84	cmp	%D05A:D0 38 06	eschar,x	;compare to the command character
85	phb	%D05D:08		;save result of comparison for a bit
86	ldx	%D05E:AE F8 06	temp	;load x= index to cmd's first chr
87	plp	%D061:28		;retrieve result of comparison of char to command char
88 ;				
89	beq	%D062:F0 13	flag1	;yes this 1-char cmd followed by nother cmd
90	cmp	%D064:5C9 0D	4charCR	;is it a (guess what) CR?
91	beq	%D066:F0 17	oneletter	;yes - a 1-char command
93 ;	come here for	%D068:		unimplemented but legal 2-char commands
95	cm2null	%D068	*	
96	plx	%D068:FA		;pull x (Cn) off stack
97	lda	%D069:AD 79 06	oldcur	;restore non-cmd-mode cursor
98	sta	%D06C:8D F8 07	cursor	
99	asl	%D06E:1E B8 03	sermode,x	;clear cmd-mode bit (bit 7 of sermode)
100	lsl	%D072:5E B8 03	sermode,x	;by shifting out bit 7 & shifting in a 0
101	bra	%D075:80 A8	nocmd	;return marking character not handled
103	flag1	%D077:	*	;come here if get eschar after LXF or T
104	plx	%D077:FA		;need X=Cn to set bit 0 of sermode
105	phx	%D078:DA		;but leave Cn on stack too
106	lnc	%D079:FE B8 03	sermode,x	;bit 0 was 0, but is now 1
107 ;				1 means new command mode
108	ldx	%D07C:AE F8 06	temp	;X= index to cmd's first chr
109	oneletter	%D07F:	*	;come here if 2-char cmd turns out 1 chr
110	lda	%D07F:BD D5 22	cm2list,x	;get command chr
111	bra	%D082:80 0B	backtol	;treat it as if we just got it
113	incmd1	%D084:	*	;in command mode, not 2-chrs tho
114	phx	%D084:DA		;save slot
115	ldx	%D085:A2 04	#4	;check 5 possible 2-chr cmds
116	cm2loop	%D087:D0 25 D2	cm2list,x	;is it there?
117	beq	%D088:F0 71	cm2found	;yes, need to flag it for next time

85	pmp		.save result of comparison to a bit
86	ldx	#003D:0E F8 06	:reload X= index to cmd's first chr
87	temp		:retrieve result of comparison of char
88	plp	#0061:28	:to command char
89	beq	flagit	:yes tis i-char and followed by nother cmd
90	cmp	#charC	:is it a (guess what) CR?
91	beq	oneLetter	:yes - a i-char command
92			
93			:come here for unimplemented but legal 2-char commands
94			
95			
96			
97			
98			
99			

D068		95 cmd2null	equ	*	;pull x (Cn) off stack ;restore non-cmd-mode cur
D069	D068:FA	96	p1x	ldx	
D070	D068:AD 79 06	97	p1a	oldcwr	
D071	D06C:8D F8 07	98	sta	cursor	
D072	D06F:1E B8 03	99	asl	sermode,x	
D073	D072:5E B8 03	100	lsl	sermode,x	
D074	D075:80 A8 D01F	101	bra	nocwd	
D075					
D076					
D077		103 flaglit	equ	*	;come here if get eschar ;need X=Cn to set bit 0
D078	D077:FA	104	p1x	ldx	
D079	D078:DA	105	p1x	ldx	
D080	D079:FE B8 03	106	lnc	sermode,x	
D081	D07C:	107 :			
D082	D07C:AE F8 06	108	ldx	* temp	
D083	D07C:	109 oneletter	equ	*	;X= index to cmd's first ;come here if 2-chr cmd t
D084	D07F:BD 25 D2	D07F	lda	cmd2llist,x	
D085	D082:80 0B D08F	D08F	bra	backto!	
D086					
D087					
D088					
D089	D084:	D084	incwdl	*	;in command mode, not 2-c
D090	D084:DA	114	p1x		;save slot
D091	D085:A2 04	115	ldx	#4	;check 5 possible 2-chr c
D092	D087:D0 25 D2	116 cmd2loop	cmp	cmd2llist,x	;is it there?
D093	D08A:F0 71 D0FD	117	beq	cmd2found	;yes, need to flag it for

D0B8:4C	39 D1		176 xready	jmp	cmdi	:go do mask stuff to RMGS
D0B8:			176 ;sermode bit 0 tells whether to set or clear command mode			
D0DE:						
D0DE:	D0DE:	180 cdone	equ	*	sermode,x	;so get it
D0DE:BD	B8 03	181 lda	lda	A	sermode,x	;shift bit 0 to carry
D0E1:4A		182 bcs	bcs	cominit1		;if set, start new cmd mode
D0E2:B0 D1	D0B5	183 bcs	bcs	oldcur		;Restore the cursor
D0E4:BD	79 06	184 lda	lda	sta	cursor	;e fall through to cmet with carry clear
D0F4:BD	F8 07	185 sta	sta	phip	sermode,x	;set command mode according to carry
D0FA:A8		186 cmet		asl	sermode,x	
D0E3:IE	B8 03	187 asl	asl	rdr	sermode,x	;leaves carry clear
D0E3:28		188 plp	plp	rdr	sermode,x	;character handled
D0F7:F8	B8 03	189 ror	ror	pls	sermode,x	;because carry clear....
D0F2:68		190 pla	pla	rts		
D0F3:60		191 rts	rts			
D0F4:		193 cmd21	equ	*		;come here to handle IE & LD
D0F4:A9	4C	194 lda	lda	#54C		;make IE look like L
D0F5:28		195 pld	pld	backtol		;get p back with carry indicating E or D
D0F7:S0	96	196 bcs	bcs	#54B		;carry set means it was an E
D0F9:A9	4B	197 lda	lda	backtol		;make LD look like X
D0FB:80	92	198 bra	bra			
D0FD:8A		200 cmd2found	ta			;copy index of cmd to acc
D0FE:FA		201 plx	plx			;restore X to Cn
D0FF:DD	B8 03	202 ora	ora	sermode,x		;copy top 2 bits of sermode
D102:09	08	203 ora	ora	#508		;e set bit 3 - 2-chr-command-mode flag
D104:9D	B8 03	204 sta	sta	sermode,x		;set sermode = index to 2-chr cmds issued
D107:38		205 sec	sec			;set carry so we stay in command mode
D108:E0	E0	206 bra	bra	cmet		;for next time
D10A:A9	D1	208 cmfound	lda	#cmdcr		;get hi byte of where to go
D10C:48		209 pha	pha			;save it on stack
D10D:BD	F5 D1	210 lda	lda	cmtable,x		;get lo byte of where to go
D110:48		211 pha	pha			;save it on stack
D111:60		212 rts	rts			;go there by Rtsing
D112:28		214 cmd,c	plp			;restore status to check carry bit
D113:FA		215 plx	plx			;restore slot number in x
D114:B0	05	216 bcs	bcs	cmd.c1		;skip if enable
D116:9E	B8 04	217 stz	stz	pwth,x		;CD is same as PMWTH=0, no CR
D119:80	C3	218 bra	bra	cdone		;we're done here
D11B:BC	86 D1	220 cmd,c1	ldy	defidx2-\$Cl,x		;get y index into aux screenholes
D11E:20	2A D2	221 jsr	jsr	r,getcrl		;go get it from aux
D121:9D	B8 04	222 sta	sta	pwth,x		;restore default PMWTH
D124:80	B8	223 bra	bra	cdone		;we're done here
D126:FA		225 cmdz	plx			;zero escape character
D127:9E	B8 04	226 stz	stz	pwth,x		;And the width
D12A:A9	00	227 lda	lda	#0		
D12C:4C	A2 D0	228 jmp	jmp	cmdz2		
D12F:		230 cmdcr	equ	*		
D12F:2F:		231 cmdn	equ	*		

```

D12F:7A      232      ply      number      ;Get number inputted
D130:AD 7E 07      233      lda          ;skip if 0
D133:F0 05 D13A      234      beq          ;Update printer width
D135:99 B8 04      235      sta      pwidth,y
D138:F0      236      dfb          ;REQ opcode to skip next byte (the PLX)
D139:      237      cmd1      equ      *
D139:      238      cmdk      equ      *
D139:      239      cmd1      equ      *
D139:7A      240      ply      flags,y
D13A:B9 B8 06      241      cmd12      ;Mask off bit we'll change
D13D:3D 02 D2      242      and      mask1,x
D140:1D 0D D2      243      ora      mask2,x
D143:99 B8 06      244      sta      flags,y
D146:98      245      tya
D147:AA      246      tax
D148:4C DE D0      247      cbne2      jmp      cbne

D148:88      249      cmdp      dey
D14C:A9 1F      250      cmdd      ;Make y point to command reg
D14E:38      251      sec          ;Mask off high three bits
D14F:90      252      dfb          ;C-1 means high 3 bits
D150:A9 F0      253      cmdb      ;RCC opcode to skip next byte
D152:18      254      cbc          ;Mask off lower 4 bits F0 = BHE
D153:39 F3 BF      255      and      scntl,y
D156:8D F8 06      256      sta      temp
D159:FA      257      plx
D15A:AD 7E 07      258      lda      number
D15D:29 0F      259      and      #0F
D15F:90 05 D166      260      bcc      noshift
D161:0A      261      asl      A
D162:0A      262      asl      A
D163:0A      263      asl      A
D164:0A      264      asl      A
D165:0A      265      asl      A
D166:0D F8 06      266      noshift      ora      temp
D169:C8      267      iny
D16A:80 17 D183      268      bra      cmdp2

D16C:B9 FA BF      270      cmds      lda      scmd,y
D16F:48      271      pha
D170:09 0C      272      ora
D172:99 FA BF      273      sta      scmd,y
D175:A9 E9      274      lda
D177:A2 53      275      mswait      ldx
D179:48      276      msloop      pha
D17A:68      277      pla
D17B:CA      278      dex
D17C:0D FB      279      bne      msloop
D17E:3A      280      dec      a
D17F:D0 F6 D177      281      bne      mswait
D181:68      282      pla
D182:FA      283      plx
D183:      284      cmdp2      equ      *
D183:99 FA BF      285      sta      scmd,y
D186:80 C0 D148      286      bra      cbne2

```

```

D188:      288      cmdr      equ      *
D188:99 F9 BF      289      sta      stat,y
D18B:AD 7E 06      290      lda      vfacty
D18E:0A      291      asl
D18F:20      292      jsr      swchk2
D192:90 03 D197      293      bcc      cmdq
D194:2D 90 C7      294      jsr      swzqt2
D197:18      295      cmdq      cbc
D198:80      296      dfb      $80
D199:38      297      cmdt      sec
D19A:FA      298      plx
D19B:2D 30 D1      299      jsr      setterm
D19E:80 A8 D148      300      bra      cbne2

D1A0:      302      setterm      equ      *
D1A0:8D B8 03      303      lda      sermode,x
D1A3:89 40      304      bit      $F40
D1A5:90 12 D1B9      305      bcc      stclr
D1A7:D0 20 D1C9      306      bne      stwasok
D1A9:E4 39      307      cpx      ksw
D1AB:D0 47 D1F4      308      bne      strts
D1AD:09 40      309      ora
D1AF:AC F9 06      310      ldy      oldcur
D1B2:8C 7A 06      311      sty      oldcur2
D1B5:A0 DF      312      ldy      #termcur
D1B7:80 07 D1C0      313      bra      stset
D1B9:F0 0E D1C9      314      stclr
D1BB:29 BF      315      and      #8F
D1BD:AC 7A 06      316      lda      oldcur2
D1C0:9D B8 03      317      stset
D1C3:8C 79 06      318      sty      oldcur
D1C6:8C F8 07      319      sty      cursor
D1C9:8C 42 C1      320      stwasok      ldy      demo2,x
D1CC:58      321      cli
D1CD:08      322      php
D1CE:78      323      sel
D1CF:B9 FA BF      324      lda      scmd,y
D1D2:09 02      325      ora
D1D4:90 02 D1D8      326      bcc      cmdt2
D1D6:29 FD      327      equ      *
D1D8:      328      cmdt2      equ      scmd,y
D1D8:99 FA BF      329      sta      scmd,y
D1D8:A9 00      330      lda      #0
D1D2:6A      331      ror      a
D1D2:80 FA 05      332      sta      typhe
D1E1:10 07 D1EA      333      bpl      cmdt3
D1E3:9C 7C 05      334      stz      twser
D1E6:9C 7C 06      335      stz      trser
D1E9:8A      336      txa
D1EA:8D FC 04      337      cmdt3      sta      aclabuf
D1ED:28      338      pip
D1EE:8E FC 05      339      flush      stx      twkey
D1F1:8E FE 06      340      stx      trkey
D1F4:60      341      strts
D1F5:      343      MSB      OFF

```

```

D400: 3 *****
D400: 4 * BASICIN - input from basic
D400: 5 *
D400: 6 * creates +XXXX, +YYYY, +SS
D400: 7 * XXXX = x position, YYYY = y position, SS = status
D400: 8 * - = key pressed
D400: 9 * 1 = button pressed
D400: 10 * 2 = button just pressed
D400: 11 * 3 = button just released
D400: 12 * 4 = button not pressed
D400: 13 *****
D400: 15 basicin sta (basl),y ;fix flashing char
D400: 16 lda p>inenc ;fix input entry
D400: 17 sta kswl
D400: 18 lda kbd ;test the keyboard
D400: 19 asl A ;save kbd and int stat for later
D400: 20 plp ;no interrupts while getting position
D400: 21 sei ;move x position into the buffer
D400: 22 jsr x.read
D400: 23 ldy #5
D400: 24 ldx mouh
D400: 25 lda mouh
D400: 26 jsr hexdec
D400: 27 ldy #12
D400: 28 ldx mouh
D400: 29 lda mouh
D400: 30 jsr hexdec
D400: 31 lda moustat
D400: 32 rol A
D400: 33 rol A
D400: 34 rol A
D400: 35 and #3
D400: 36 eor #3
D400: 37 inc A
D400: 38 plp
D400: 39 ldy #16
D400: 40 jsr hexdec2
D400: 41 ply
D400: 42 ldx #17
D400: 43 lda #80
D400: 44 putinbuf sta inbuf,x
D400: 45 jmp swtsc2 ;goback
D400: 47 *****
D400: 48 * HEXTODEC - puts +0000, into the input buffer
D400: 49 * inputs: a = low byte of number
D400: 50 * x = high byte of number
D400: 51 * y = position of ones digit
D400: 52 *****
D400: 54 hexdec cpx #80 ;is it a negative number?
D400: 55 bcc hexdec2
D400: 56 eor $FF ;form two's complement
D400: 57 adc #0 ;c = 1 from compare
D400: 58 pha ;save it
D400: 59 txa
D400: 60 eor $FF

```

```

D1F5: 344 cndtable equ *
D1F5: 345 ddb >cndi-1 ;command routines' lo bytes
D1F6: 346 ddb >cndi-1
D1F7: 347 ddb >cndi-1
D1F8: 348 ddb >cndi-1
D1F9: 349 ddb >cndi-1
D1FA: 350 ddb >cndi-1
D1FB: 351 ddb >cndi-1
D1FC: 352 ddb >cndi-1
D1FD: 353 ddb >cndi-1
D1FE: 354 ddb >cndi-1
D1FF: 355 ddb >cndi-1
D200: 356 ddb >cndi-1
D201: 357 ddb >cndi-1
D202: 358 *****
D202: 359 * masks for: I K L N CR XE XD FE FD ME MD
D202: 360 mask1 ddb $7F,$BF,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F,$7F
D200: 361 mask2 ddb $80,$80,$40,$80,$80,$40,$80,$80,$40,$80,$80,$40,$80,$80,$40,$80
D218: 363 cndlist equ *
D218: 364 asc "IKLM"
D21C: 365 ddb $0D ;cr (part of cndlist)
D21D: 366 asc "BDPQRST"
D225: 367 cnd2list equ *
D225: 368 asc "LXMC" ;2-chr commands' first chrs
D22A: 370 *****
D22A: 371 * R.GETALT is the same as GETALT in main rom. Only the
D22A: 372 * location is different.
D22A: 373 *****
D22A: 375 r.getalt lda rdramrd ;save state of aux memory
D22D: 376 asl
D22E: 377 lda rd80col ;and the 80STORE switch
D231: 378 plp
D232: 379 sta clr80col ;no 80STORE to get page 1
D235: 380 sta rdcardram ;pop in the other half of RAM
D236: 381 lda $478,y ;read the desired byte
D238: 382 plp ;and restore memory
D23C: 383 bcs r.getalt1
D23E: 384 sta rdmainram
D241: 385 r.getalt1 bpl r.getalt2
D243: 386 sta set80col
D246: 387 r.getalt2 rts
D247: 389 defidx2 ddb 3,7 ;same as DEFIDX in main rom.
D249: 72 include mbasic
D249: 01B7 1 ds $0400-*,0

```

```

D44D:69 00      61      adc      #0
D44F:7A         62      tax
D450:68         63      pla
D451:38         64      sec
D452:8D 14 02   65      sta      binl
D453:8E 15 02   66      stx      binh
D458:A9 2B      67      lda      #'+'
D45A:90 02      68      bcc      hdp0s2
D45C:A9 2D      69      lda      #'-'
D45E:48         70      pha
D45F:A9 2C      71      lda      #'.'
D461:99 01 02   72      sta      inbuf+1,y
D464:          73      hdl0op      equ      *

D464:          75      * divide BINH,L by 10 and leave remainder in a
D464:A2 11      77      ldx      #16+1
D466:A9 00      78      lda      #0
D468:18         79      clc
D469:2A         80      div10loop
D46A:C9 0A      81      cmp      #10
D46C:90 02      82      bcc      div10lt
D46E:89 0A      83      sbc      #10
D470:2E 14 02   84      div10lt
D473:2E 15 02   85      rol      binh
D476:CA         86      dex
D477:D0 F0      87      bne      div10loop
D479:09 30      88      ora      #'0'
D47B:99 00 02   89      sta      inbuf,y
D47E:88         90      dey
D47F:F0 08      91      beq      hddone
D481:C0 07      92      cpy      #7
D483:F0 04      93      beq      hddone
D485:C0 0E      94      cpy      #14
D487:D0 0B      95      bne      hdl0op
D489:68         96      hddone
D48A:99 00 02   97      sta      inbuf,y
D48D:60         98      rts
D48E:          73      include banger

```

```

D48E:          3 * These routines test all 128K ram. All combinations of soft
D48E:          4 * switches applicable to the //c are tested and verified.
D48E:          5 *
D48E:          6 * In the event of any failure, the diagnostic is halted. A message
D48E:          7 * is written to screen memory indicating the source of the failure.
D48E:          8 * When RAM fails the message is composed of "RAM ZP" (indicating
D48E:          9 * failure detected in the first page of RAM) or "RAM" (meaning the
D48E:         10 * other 63.75K), followed by a binary representation of the failing
D48E:         11 * bits set to "1". For example, "RAM 0 1 1 0 0 0 0 0" indicates
D48E:         12 * that bits 5 and 6 were detected as failing. To represent
D48E:         13 * auxiliary memory, a "*" symbol is printed preceding the message.
D48E:         14 *
D48E:         15 * When the MMU or IOU fail, the message is simply "MMU" or "IOU".
D48E:         16 * If the IOUDIS or DRINES switch fails, the message is "GIU".
D48E:         17 *
D48E:         18 * The test will run continuously for as long as the Open and Closed
D48E:         19 * Apple keys remain depressed for no keyboard is connected) and no
D48E:         20 * failures are encountered. The message "System OK" will appear in
D48E:         21 * the middle of the screen when a successful cycle has been run and
D48E:         22 * either of the Apple keys are no longer depressed. Another cycle
D48E:         23 * may be initiated by pressing both Apple keys again while this
D48E:         24 * message is on the screen. To exit diagnostics, Control-Reset
D48E:         25 * must be pressed without the Apple keys depressed.
D48E:         26 *
D48E:         27 *
D48E:         0011 28 GIUDIX EQU $11
D48E:         0009 29 IOUDIX EQU $09
D48E:         0001 30 MMUIDX EQU $01
D48E:         0588 31 SCREEN EQU $588
D48E:         32 *
D48E:         33 DIAGS sta ktxclr ;text mode off
D48E:         34 sta loudshl ;Disable IOU
D48E:         35 sta setan3 ;Double hires off

D497:          37 * Test Zero-Page, then all of memory. Report errors when
D497:          38 * encountered. Accumulator can be anything on entry. All
D497:          39 * registers used, but no stack. Addresses between $C000
D497:          40 * and $CFFF are mapped to main $0000 bank. Addresses
D497:          41 * between $C000 and $CFFF are mapped to main $0000 bank.

D497:A0 04      43      TSTZPG ldy #54
D499:A2 00      44      ldx #0
D49B:18         45      clc
D49C:79 2A C8   46      adc      ntbl,y
D49F:95 00      47      sta      $00,x
D4A1:E8         48      inx
D4A2:D0 F7      49      bne      zp1
D4A4:18         50      clc
D4A5:79 2A C8   51      adc      ntbl,y
D4A8:D5 00      52      cmp      $00,x
D4AA:D0 10      53      bne      ZPERORR
D4AC:E8         54      inx
D4AD:D0 F5      55      bne      zp2
D4AF:6A         56      ror      a
D4B0:2C 19 C0   57      bit      rdvblbar
D4B3:10 02      58      bpl      zp3
D4B5:49 A5      59      cor      #5A5
D4B7:88         60      dey

```

```

;fill zero page with a pattern
;after all bytes filled,
; ACC has original value again.
;so values can be tested
;branch if memory failed
;loop until all 256 bytes tested
;change ACC so location $FF will change
; use RDVBL for a little randomness...
;use a different pattern now

```

[illegible]

```

D510:      D5      $D600--4, $00
D600:      *****
D600:      4 * Initmouse - resets the mouse
D600:      5 * Also clears all of the mouse holes
D600:      6 * note that iou access fires pdlstrb & makes mouse happy
D600:      7 *****

          9 i.nitmouse stz moustat      ;Clear status
          10      ldx #80
          11      ldy #1
          12      stz minxl,x
          13      stz minlh,x
          14      lda #FFF
          15      sta maxxl,x
          16      lda #03
          17      sta maxlh,x
          18      ldx #0
          19      dey
          20      bpl xloop
          21      jsr x.hole
          22      lda #0

          24 *****
          25 * XSETMOU - Sets the mouse mode to A
          26 *****

          28 x.setmou tax
          29      jsr m.ovelrq
          30      tax
          31      sta mouamp
          32      lsr A
          33      ora mouamp
          34      cmp #810
          35      bcs salwinald
          36      and #5
          37      beq xsoff
          38      cli
          39      xsoff

D621:
D621:
D621:
D621:AA
D622:20 46 C7
D623:8A
D624:80
D625:8A
D626:80 78 04
D627:AA
D628:0D 78 04
D629:C9 10
D62A:20 1F
D62B:80 1F
D62C:29 05
D62D:29 01
D62E:58
D62F:58 55

```

```

D638: 41 *****
D638: 42 *
D638: 43 * SEIOU - Sets the IOU interrupt modes to A
D638: 44 * Inputs: A = Bits to change
D638: 45 * D7 = Y int on falling edge
D638: 46 * D6 = Y int on rising edge
D638: 47 * D5 = X int on falling edge
D638: 48 * D4 = X int on rising edge
D638: 49 * D3 = Enable VBL int
D638: 50 * D2 = Disable VBL int
D638: 51 * D1 = Enable mouse int
D638: 52 * D0 = Disable mouse int
D638: 53 *
D638: 54 *****

D638: 56 setiou    php
D638: 57          sei
D638: 58          stx    moumode
D638: 59          sta    iouenbl

;Don't allow ints while iou enabled
;Enable iou access

```

33 MOUSEINT.X	Apple //c Diagnostics	20-OCT-86 06:41 PAGE 131
D640:32 08	60 ldx #8	
D642:CA	61 siloop dex	
D643:0A	62 asl A	;Get a bit to check
D644:90 03 D649	63 bcc sinoch	;No change if C=0
D646:90 58 C0	64 sta iou.x	;Set it
D649:00 F7 D642	65 sinoch bne siloop	;Any bits left in A?
D648:80 78 C0	66 sta loudshl	;Turn off iou access
D64E:28	67 plp	
D64F:18	68 clc	
D650:60	69 sminvalid rts	
D651:	71 *****	
D651:	72 * XMCLEAR - Clears mouse position & status	
D651:	73 *****	
D651:A2 80	75 x.mhome ldx #80	;Point mouse to upper left
D653:80 02 D657	76 bra xmh2	
D655:A2 00	77 xhloop ldx #0	
D657:80 7D 04	78 xmh2 lda minl,x	
D65A:90 7F 04	79 sta mouxl,x	
D65D:80 7D 05 80	80 lda minlh,x	
D660:90 7F 05	81 sta moudh,x	
D663:CA	82 dex	
D664:10 EF D655	83 bpl xhloop	
D666:80 0C D674	84 bra xcdone	
D668:	86 *****	
D668:	87 * XMCLEAR - Sets the mouse to 0,0	
D668:	88 *****	
D668:9C 7F 04	90 x.mclear stz mouxl	
D66B:9C 7F 05	91 stz moudh	
D66E:9C 7F 04	92 stz mouyl	
D671:9C 7F 05	93 stz moudh	
D674:9C 7F 06	94 xcdone stz mounam	
D677:18	95 clc	
D678:60	96 rts	
D679:	98 *****	
D679:	99 * XMRAD - Updates the screen holes	
D679:	100 *****	
D679:A9 20	102 x.mread lda #mouarm	;Has mouse moved?
D67B:1C 7F 07	103 trb moustat	;Clear moved bit in stat
D67E:2D 7F 06	104 and mounam	
D681:1C 7F 06	105 trb mounam	;Clear arm bit
D684:2C FF 07	106 bit mounmode	;If D7 = 1 leave buttons alone
D687:30 13 D69C	107 bml xmr2	
D689:2C 63 C0	108 bit moubut	
D68C:30 02 D690	109 bml xrbt	;Button pressed?
D68E:09 80	110 ora #80	
D690:2C 7F 07	111 xrbt bit moustat	;Pressed last time?
D693:10 02 D697	112 bpl xrbt2	
D695:09 40	113 ora #80	
D697:80 7F 07	114 xrbt2 sta moustat	
D69A:18	115 clc	
D69B:60	116 rts	
D69C:	117 xmr2 equ *	;Leave button bits alone
33 MOUSEINT.X	Apple //c Diagnostics	20-OCT-86 06:41 PAGE 132
D69C:0D 7F 07	118 ora moustat	
D69F:29 20	119 and #80	;button bits
D6A1:80 F4 D697	120 bra xrbt2	
D6A3:	122 *****	
D6A3:	123 * XMCAMP - Store new bounds	
D6A3:	124 * Inputs A = 1 for Y, 0 for X axis	
D6A3:	125 * minl, minh, maxl, maxh = new bounds	
D6A3:	126 *****	
D6A3:6A	128 x.mclap for A	;1 -> 80
D6A4:6A	129 ror A	
D6A5:29 80	130 and #80	
D6A7:AA	131 tax	
D6A8:AD 78 04	132 lda minl	
D6AB:9D 7D 04	133 sta minl,x	
D6AE:AD 78 05	134 lda minh	
D6B1:9D 7D 05	135 sta minh,x	
D6B4:AD 78 04	136 lda maxl	
D6B7:9D 7D 06	137 sta maxl,x	
D6BA:AD 78 05	138 lda maxh	
D6BD:9D 7D 07	139 sta maxh,x	
D6C0:18	140 clc	;No error
D6C1:60	141 rts	
D6C2:	143 *****	
D6C2:	144 * XMTSTINT - Checks mouse status bits	
D6C2:	145 * Used for user mouse interrupt	
D6C2:	146 *****	
D6C2:48	148 x.mstint pha	
D6C3:18	149 clc	
D6C4:A9 0E	150 lda #80E	
D6C6:2D 7F 07	151 and moustat	
D6C9:00 01 D6CC	152 bne nostat2	
D6CB:38	153 sec	
D6CC:68	154 nostat2 pla	
D6CD:60	155 rts	
D6CE:	0032 157 ds \$D700-*,\$800	
D700:	75 include s.execute	
D700:	0100 1 ds \$D800-*,\$800	

```

61 * returns status block for call 0
62 * 1 0 0 0 0 0 0
63 *****
65 pstat0 lda pstat ;must be call 0
66 bne stbad ;branch if bad
67 sta sval ;set bytes read count
68 ldy #8
69 sty sval
70 dey
71 st01p (gbuff),y ;save out the 0s
72 dey
73 bne st01p
74 lda #1
75 sta (gbuff),y
76 rts

78 *****
79 * PCNTL - control call
80 * PSTATUS - status call for device 1
81 * call 0 (reset) is implemented for both devices
82 *****
83 pnt01 lda pstat ;call 0?
84 beq pntok
85 stbad lda #badctl ;oops! bad status/control number
86 sty error
87 pntok rts

89 *****
90 * PSTATUS - status call for device 1
91 * call 0,3 supported
92 *****
94 sl.pstat0 lda #4 ;number bytes for call 0
95 ldx pstat
96 beq pst0
97 cpz #3 ;is it #3?
98 bne stbad ;branch if bad call
99 lda #25 ;# bytes for call 3
100 sta sval
101 ldx #0
102 sty sval
103 tay
104 dey
105 pstmov lda stattbl,y ;move the status info
106 sta (gbuff),y
107 dey
108 bpl pstmov
109 ldy sl.mslot ;get the size
110 ldx numbanks,y
111 lsr A
112 ldy #2
113 sta (gbuff),y
114 rts

116 *****
117 * XREAD - read a block
118 * XWRITE - write a block

```

```

D853:
D854:
D855:
D856:
D857:
D858:
D859:
D860:
D861:
D862:
D863:
D864:
D865:
D866:
D867:
D868:
D869:
D870:
D871:
D872:
D873:
D874:
D875:
D876:
D877:
D878:
D879:
D880:
D881:
D882:
D883:
D884:
D885:
D886:
D887:
D888:
D889:
D890:
D891:
D892:
D893:
D894:
D895:
D896:
D897:
D898:
D899:
D900:
D901:
D902:
D903:
D904:
D905:
D906:
D907:
D908:
D909:
D910:
D911:
D912:
D913:
D914:
D915:
D916:
D917:
D918:
D919:
D920:
D921:
D922:
D923:
D924:
D925:
D926:
D927:
D928:
D929:
D930:
D931:
D932:
D933:
D934:
D935:
D936:
D937:
D938:
D939:
D940:
D941:
D942:
D943:
D944:
D945:
D946:
D947:
D948:
D949:
D950:
D951:
D952:
D953:
D954:
D955:
D956:
D957:
D958:
D959:
D960:
D961:
D962:
D963:
D964:
D965:
D966:
D967:
D968:
D969:
D970:
D971:
D972:
D973:
D974:
D975:
D976:
D977:
D978:
D979:
D980:
D981:
D982:
D983:
D984:
D985:
D986:
D987:
D988:
D989:
D990:
D991:
D992:
D993:
D994:
D995:
D996:
D997:
D998:
D999:

```

34 S.EXECUTE	slinky execution routines	20-OCT-86 06:41 PAGE 135	34 S.EXECUTE	slinky execution routines	20-OCT-86 06:41 PAGE 136
D89C:	119 *		D8EF:	174 *****	
D89C:	120 * PRODOS read & write are changed into Protocol converter read block		D8EF:	175 * here is the rest of the boot code	
D89C:	121 * and write block which are then changed into read & write		D8EF:	176 * input: a = kshw, output: v = 1 if boot fails	
D89C:	122 *****		D8EF:	177 * jumps to DOS patch if IN# from DOS	
D89C:	123 *****		D8EF:	178 *****	
D89C:2C 39 D8	124 xread bit iorts		D8EF:AO C4	180 boot.sl ldy #4+\$C0	
D8A1:50	125 dfb \$50		D8F1:8C F8 07	181 sty sl.mslot	
D8A2:58	126 xwrite clv		D8F4:A2 C8	182 ldx #4+\$10:\$88	
D8A3:A5 47	128 xrwcmn lda block+1		D8F6:8E 78 07	183 stx sl.demo	
D8A5:85 48	129 lda block+1		D8F9:CD F8 07	184 cmp sl.mslot	
D8A7:A5 46	130 lda block		D8FC:D0 09 D907	185 bne btndos	
D8A9:85 47	131 sta block		D8FE:AO 00 BF	186 lda proflag	
D8AB:A5 45	132 lda block		D901:70 04 D907	187 beq btndos	
D8AD:85 46	133 lda buffer+1		D903:C9 4C	188 cmp #54C	
D8AF:A5 44	134 sta pbuffer+1		D905:30 20 D927	189 bne dospatch	
D8B1:85 45	135 sta pbuff		D907:9C 01 08	191 btndos stz bootbuf+1	
D8B3:A9 00	136 lda #0		D90A:	192 :lda power2,y ;if power up bytes not set, don't boot	
D8B5:85 49	137 sta pblock+2		D90A:	193 :eor #5A5	
D8B7:F0 05 D8BE	138 beq xreadz		D90A:	194 :cmp powerup,y	
D8B9:	140 *****		D90A:89 B8 06	195 lda powerup,y	
D8B9:	141 * PROBLK - Protocol converter block read		D90D:C9 A5	196 cmp #5A5	
D8B9:	142 * PROBLK - Protocol converter block write		D90F:D0 11 D922	197 bne btfail	
D8B9:	143 *****		D911:A0 03	198 ldy #3	
D8B9:2C 39 D8	145 prdblk bit iorts		D913:B9 23 D9	199 btuv lda btcmd,y	
D8BC:50	146 dfb \$50		D916:99 44 00	200 sta buffer,y	
D8BD:88	147 prblk clv		D919:88	201 dey	
D8BE:A5 47	149 xread2 lda pblock		D91A:10 F7 D913	202 bpl btuv	
D8C0:0A	150 asl A		D91C:AC F8 07	203 ldy sl.mslot	
D8C1:85 4A	151 sta paddr+1		D91F:20 9E D8	204 jsr xread	
D8C3:A5 48	152 lda pblock+1		D922:60	205 btfail rts	
D8C5:2A	153 rol A		D923:00 08	207 btcmd dw \$800	
D8C6:85 4B	154 sta paddr+2		D925:00 00	208 dw 0	
D8C8:80 1F D8E9	155 bcs prbad2		D927:	210 *****	
D8CA:A5 49	156 lda pblock+2		D927:	211 * DOSPATCH - patches rws to jump to us	
D8CC:D0 1B D8E9	157 bne prbad2		D927:	212 *****	
D8CE:85 49	158 sta paddr		D927:A9 4C	214 dospatch lda #54C	
D8D0:85 47	159 sta pcount		D929:8D 00 BD	215 sta rws	
D8D2:A9 02	160 lda #2		D92C:A9 D1	216 lda #dosent4	
D8D4:85 48	161 sta pcount+1		D92E:8D 01 BD	217 sta rws+1	
D8D6:AD 14 C0	162 lda rdramwt		D931:8C 02 BD	218 sty rws+2	
D8D9:70 03 D8DE	163 bvs prread		D934:A9 C3	219 lda #>dossyn	
D8DB:AD 13 C0	164 lda rdramd		D936:8D 1E 9D	220 sta dosinit	
D8DE:29 80	165 prthead and #80		D939:A9 A6	221 lda #<dossyn	
D8E0:05 4B	166 ora paddr+2		D93B:8D 1F 9D	222 sta dosinit+1	
D8E2:85 4B	167 sta paddr+2		D93E:8E	223 pla	
D8E4:70 06 D8EC	168 bvs prbad3		D93F:68	224 pla	
D8E6:4C F7 C5	169 jmp sl.pwrite		D940:FA	226 plx	
D8E9:4C E5 C5	170 prbad2 jmp prbad		D941:FE 00 C0	227 inc \$C000,x	
D8EC:4C 80 C5	172 prbad3 jmp sl.pread		D944:FA	228 plx	
			D945:68	229 pla	
			D946:68	230 pla	

34 S.EXECUTE	slinky execution routines	20-OCT-86 06:41 PAGE 135	34 S.EXECUTE	slinky execution routines	20-OCT-86 06:41 PAGE 136
D89C:	119 *		D8EF:	174 *****	
D89C:	120 * PRODOS read & write are changed into Protocol converter read block		D8EF:	175 * here is the rest of the boot code	
D89C:	121 * and write block which are then changed into read & write		D8EF:	176 * input: a = kshw, output: v = 1 if boot fails	
D89C:	122 *****		D8EF:	177 * jumps to DOS patch if IN# from DOS	
D89C:	123 *****		D8EF:	178 *****	
D89C:2C 39 D8	124 xread bit iorts		D8EF:AO C4	180 boot.sl ldy #4+\$C0	
D8A1:50	125 dfb \$50		D8F1:8C F8 07	181 sty sl.mslot	
D8A2:58	126 xwrite clv		D8F4:A2 C8	182 ldx #4+\$10:\$88	
D8A3:A5 47	128 xrwcmn lda block+1		D8F6:8E 78 07	183 stx sl.demo	
D8A5:85 48	129 lda block+1		D8F9:CD F8 07	184 cmp sl.mslot	
D8A7:A5 46	130 lda block		D8FC:D0 09 D907	185 bne btndos	
D8A9:85 47	131 sta block		D8FE:AO 00 BF	186 lda proflag	
D8AB:A5 45	132 lda block		D901:70 04 D907	187 beq btndos	
D8AD:85 46	133 lda buffer+1		D903:C9 4C	188 cmp #54C	
D8AF:A5 44	134 sta pbuffer+1		D905:30 20 D927	189 bne dospatch	
D8B1:85 45	135 sta pbuff		D907:9C 01 08	191 btndos stz bootbuf+1	
D8B3:A9 00	136 lda #0		D90A:	192 :lda power2,y ;if power up bytes not set, don't boot	
D8B5:85 49	137 sta pblock+2		D90A:	193 :eor #5A5	
D8B7:F0 05 D8BE	138 beq xreadz		D90A:	194 :cmp powerup,y	
D8B9:	140 *****		D90A:89 B8 06	195 lda powerup,y	
D8B9:	141 * PROBLK - Protocol converter block read		D90D:C9 A5	196 cmp #5A5	
D8B9:	142 * PROBLK - Protocol converter block write		D90F:D0 11 D922	197 bne btfail	
D8B9:	143 *****		D911:A0 03	198 ldy #3	
D8B9:2C 39 D8	145 prdblk bit iorts		D913:B9 23 D9	199 btuv lda btcmd,y	
D8BC:50	146 dfb \$50		D916:99 44 00	200 sta buffer,y	
D8BD:88	147 prblk clv		D919:88	201 dey	
D8BE:A5 47	149 xread2 lda pblock		D91A:10 F7 D913	202 bpl btuv	
D8C0:0A	150 asl A		D91C:AC F8 07	203 ldy sl.mslot	
D8C1:85 4A	151 sta paddr+1		D91F:20 9E D8	204 jsr xread	
D8C3:A5 48	152 lda pblock+1		D922:60	205 btfail rts	
D8C5:2A	153 rol A		D923:00 08	207 btcmd dw \$800	
D8C6:85 4B	154 sta paddr+2		D925:00 00	208 dw 0	
D8C8:80 1F D8E9	155 bcs prbad2		D927:	210 *****	
D8CA:A5 49	156 lda pblock+2		D927:	211 * DOSPATCH - patches rws to jump to us	
D8CC:D0 1B D8E9	157 bne prbad2		D927:	212 *****	
D8CE:85 49	158 sta paddr		D927:A9 4C	214 dospatch lda #54C	
D8D0:85 47	159 sta pcount		D929:8D 00 BD	215 sta rws	
D8D2:A9 02	160 lda #2		D92C:A9 D1	216 lda #dosent4	
D8D4:85 48	161 sta pcount+1		D92E:8D 01 BD	217 sta rws+1	
D8D6:AD 14 C0	162 lda rdramwt		D931:8C 02 BD	218 sty rws+2	
D8D9:70 03 D8DE	163 bvs prread		D934:A9 C3	219 lda #>dossyn	
D8DB:AD 13 C0	164 lda rdramd		D936:8D 1E 9D	220 sta dosinit	
D8DE:29 80	165 prthead and #80		D939:A9 A6	221 lda #<dossyn	
D8E0:05 4B	166 ora paddr+2		D93B:8D 1F 9D	222 sta dosinit+1	
D8E2:85 4B	167 sta paddr+2		D93E:8E	223 pla	
D8E4:70 06 D8EC	168 bvs prbad3		D93F:68	224 pla	
D8E6:4C F7 C5	169 jmp sl.pwrite		D940:FA	226 plx	
D8E9:4C E5 C5	170 prbad2 jmp prbad		D941:FE 00 C0	227 inc \$C000,x	
D8EC:4C 80 C5	172 prbad3 jmp sl.pread		D944:FA	228 plx	
			D945:68	229 pla	
			D946:68	230 pla	

## slinky execution routines

### 34 S. EXECUTE

```

D94E:
238 *****
239 * DOSCONV - changes DOS command into ours
D94F:
240 * output: command table in zp Y = command
D94E:
241 *****
242 *****
243 dosconv ldy #ibdrvm ;get drive 1 or 2
244 lda (ibopl),y
245 cmp #1 ;only 1 valid
246 beq dci
247 jmp pzcmd ;bad drive number
D95A:
248 lda (ibopl),y ;get track & sector
249 d95C:B1 48 ;addr = 00000FFF FFFFFFFF
250 lsr A
251 ror paddr+1
252 lsr A
253 ror paddr+1
254 lsr A
255 sta paddr+2
256 lda paddr+1
257 ror A
258 and #$E0
259 iny
260 ora (ibopl),y ;or in sector
261 sta paddr+1
262 ldy #ibdrvm ;get pointer to user's buffer
263 lda (ibopl),y
264 sta pbuff
265 iny
266 ldy (ibopl),y
267 lda pbuff+1
268 d97C:A0 0C ;get command
269 d97E:B1 48 ;get track & sector
D94E:
270 beq dcrt ;0 = null = do nothing
271 and #3
272 beq dcerr ;4 = format is an error
273 ora #17 ;1 -> 17, 2 -> 19
274 tay ;Y = command
275 ldx #0 ;count = $100 bytes
276 stx pcount
277 stx paddr
278 lnx
279 stx pcount+1
280 jmp exec2
D995:
281 include s.makecat
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

## slinky execution routines

### 34 S. EXECUTE

D947:A2 00	232	ldx	#0	
D949:A9 98	233	lda	#98	;return a control-X
D94B:4C 04 C7	234	jmp	swrts2	;switch rom bank and return
D94E:60	236	dcrts	rts	

```

35 S.MAKECAT      20-OCT-86 06:41 PAGE 139
slinky miscellaneous routines
D995: 2 *****
D995: 3 * TESTSIZE - determines ramdisk size nondestructively
D995: 4 * Inputs: Y = mslot X = devno
D995: 5 *****
D995: 6 testsize equ *
D995: 7
D995: 8 sta addr1,x
D995: 9 sta addrn,x
D995: 10 lda #510
D995: 11 sec #1
D995: 12 tslot
D995: 13 sta addr1,x
D995: 14 lda data,x
D995: 15 pha
D995: 16 dec addr1,x
D995: 17 lda #A5
D995: 18 sta data,x
D995: 19 dec addr1,x
D995: 20 eor data,x
D995: 21 dec addr1,x
D995: 22 cmp #1
D995: 23 pla
D995: 24 sta data,x
D995: 25 lda addr1,x
D995: 26 and #50F
D995: 27 beq tsnoram
D995: 28 bcs tslot
D995: 29 adc #1
D995: 30 tsnoram sta numbanks,y
D995: 31 lsr A
D995: 32 sta sizetemp
D995: 33 rts

D903: 35 *****
D903: 36 * Routines for making a directory on the disk
D903: 37 *****
D903: 39 *****
D903: 40 * Format - Tests for powerup and puts catalog on the disk if needed
D903: 41 * Inputs: Y = mslot X = devno
D903: 42 *****
D903: 43 sl.format equ *
D903: 44
D903: 45 lda power2,y ;is power2 eor powerup = A5
D903: 46 eor #A5
D903: 47 cmp powerup,y
D903: 48 beq fntdone
D903: 49 sta powerup,y
D903: 50 lda #A5
D903: 51 cmp powerup,y
D903: 52 beq fntdone
D903: 53 sta powerup,y
D903: 54
D903: 55 * If all screen holes spaces, someone might have cleared
D903: 56 * the screen wrong so we won't reformat
D903: 57 cmp #505
D903: 58 fannosp equ *
D903: 59

35 S.MAKECAT      20-OCT-86 06:41 PAGE 140
slinky miscellaneous routines
D9E3:20 95 D9 60 jsr testsize
D9E6:28 61 pip
D9E7:F0 IF DA08 62 beq fntdone
D9E9:A0 00 BF 63 lda proflag
D9EC:F0 1B DA09 64 beq fmpas
D9EE:C3 4C 65 cmp #54C
D9F0:D0 1D DA0F 66 bne fmdos
D9F2:A0 FF 67 ldy #procat
D9F4:20 39 DA 68 jsr makecat
D9F7:A9 01 69 lda #01
D9F9:A0 20 70 fmpmapl ldy #32
D9F9:F0 BF 71 fmpmapl sta data,x
D9FE:09 FF 72 cmp #FFF
DA00:88 73 dey
DA01:D0 F8 D9FB 74 bne fmpmap2
DA03:C3 78 04 75 dec sizetemp
DA06:D0 F1 D9F9 76 bne fmpmapl
DA08:60 77 fntdone rts

D909: 79 * Do a Pascal catalog
D909: 80 fmpas equ *
D909:A0 78 81 ldy #pascal
D90B:20 39 DA 82 jsr makecat
DA0E:60 83 rts

D90F: 85 * Do a DOS catalog
D90F: 86 fmdos equ *
D90F:A0 2C DA0F 87 ldy #doscat
DA11:20 39 DA 88 jsr makecat
DA14:A9 44 89 lda #544
DA16:9D F8 BF 90 sta addr1,x
DA19:A0 78 04 91 lda sizetemp
DA1C:A0 72 92 ldy #114
DA1E:C9 04 93 cmp #4
DA20:90 02 DA24 94 bit fndmap
DA22:A0 BA 95 ldy #186
DA24:8D F8 BF 96 fndmapl lda addr1,x
DA27:C9 7C 97 cmp #57C
DA29:D0 05 DA30 98 bne fmdok
DA2B:A9 7E 99 lda #57E
DA2D:9D F8 BF 100 sta addr1,x
DA30:A9 FF 101 fmdok sta data,x
DA32:9D F8 BF 102 dey
DA35:88 103 bne fndmap
DA36:D0 3C DA24 104
DA38:60 105 rts

D903: 35 *****
D903: 36 * Routines for making a directory on the disk
D903: 37 *****
D903: 39 *****
D903: 40 * Format - Tests for powerup and puts catalog on the disk if needed
D903: 41 * Inputs: Y = mslot X = devno
D903: 42 *****
D903: 43 sl.format equ *
D903: 44
D903: 45 lda power2,y ;is power2 eor powerup = A5
D903: 46 eor #A5
D903: 47 cmp powerup,y
D903: 48 beq fntdone
D903: 49 sta powerup,y
D903: 50 lda #A5
D903: 51 cmp powerup,y
D903: 52 beq fntdone
D903: 53 sta powerup,y
D903: 54
D903: 55 * If all screen holes spaces, someone might have cleared
D903: 56 * the screen wrong so we won't reformat
D903: 57 cmp #505
D903: 58 fannosp equ *
D903: 59

```

```

;What type of catalog?
;JMP if ProDOS
;Do a ProDOS catalog
;Put in all but bit map
;Blocks 0-6 busy
;32 FF's for each $100 blocks
;Rest are FF's
;Point to track 3 bitmap
;Addr1 = 0 from makecat
;Check if at least 512K
;Assume 256K
;At least $400 blocks
;Make 400K volume
;Don't free catalog
;Track $11?
;Skip first 16 sectors
;7C -> 7E so no false carry

```

```

DA39: *****
DA39: 108 * MAKECAT - Creates a catalog
DA39: 109 * Inputs: X = index into catalog tables
DA39: 110 *
DA39: 111 *****
DA39: 112 makecat equ *
DA39: 113 lda #0
DA39: 114 sta addr1,x
DA39: 115 sta addrm,x
DA39: 116 sta addrh,x
DA39: 117 mcbboot
DA39: 118 lda data,x
DA39: 119 and #5F0
DA39: 120 beq mcbboot
DA39: 121 lda #4
DA39: 122 sta addrm,x
DA39: 123 mcbbyte
DA39: 124 lda catbtl,y
DA39: 125 cmp #zers
DA39: 126 beq mcb0
DA39: 127 cmp #skpfe
DA39: 128 beq mcbfe
DA39: 129 cmp #sizeflg
DA39: 130 bne mcntsz
DA39: 131 lda mcntsz
DA39: 132 bne mcntnm
DA39: 133 mcntsz
DA39: 134 bne mcntnm
DA39: 135 lda SI.MSLOT
DA39: 136 eor #5F0
DA39: 137 mcntnm
DA39: 138 jmp mcbbyte
DA39: 139 mcb0
DA39: 140 lda catbtl,y
DA39: 141 beq mcbadd
DA39: 142 mcbfe
DA39: 143 lda #0
DA39: 144 sta data,x
DA39: 145 pla
DA39: 146 sec
DA39: 147 sbc mcbfe
DA39: 148 lda #0
DA39: 149 beq mcbbyte
DA39: 150 mcbadd2
DA39: 151 mcbadd
DA39: 152 bne mcbadd2
DA39: 153 iny
DA39: 154 lda catbtl,y
DA39: 155 beq mcbone
DA39: 156 sta addrm,x
DA39: 157 iny
DA39: 158 lda catbtl,y
DA39: 159 sta addrh,x
DA39: 160 jmp mcbbyte
DA39: 161 mcbone

```

```

DA39: *****
DA39: 108 * MAKECAT - Creates a catalog
DA39: 109 * Inputs: X = index into catalog tables
DA39: 110 *
DA39: 111 *****
DA39: 112 makecat equ *
DA39: 113 lda #0
DA39: 114 sta addr1,x
DA39: 115 sta addrm,x
DA39: 116 sta addrh,x
DA39: 117 mcbboot
DA39: 118 lda data,x
DA39: 119 and #5F0
DA39: 120 beq mcbboot
DA39: 121 lda #4
DA39: 122 sta addrm,x
DA39: 123 mcbbyte
DA39: 124 lda catbtl,y
DA39: 125 cmp #zers
DA39: 126 beq mcb0
DA39: 127 cmp #skpfe
DA39: 128 beq mcbfe
DA39: 129 cmp #sizeflg
DA39: 130 bne mcntsz
DA39: 131 lda mcntsz
DA39: 132 bne mcntnm
DA39: 133 mcntsz
DA39: 134 bne mcntnm
DA39: 135 lda SI.MSLOT
DA39: 136 eor #5F0
DA39: 137 mcntnm
DA39: 138 jmp mcbbyte
DA39: 139 mcb0
DA39: 140 lda catbtl,y
DA39: 141 beq mcbadd
DA39: 142 mcbfe
DA39: 143 lda #0
DA39: 144 sta data,x
DA39: 145 pla
DA39: 146 sec
DA39: 147 sbc mcbfe
DA39: 148 lda #0
DA39: 149 beq mcbbyte
DA39: 150 mcbadd2
DA39: 151 mcbadd
DA39: 152 bne mcbadd2
DA39: 153 iny
DA39: 154 lda catbtl,y
DA39: 155 beq mcbone
DA39: 156 sta addrm,x
DA39: 157 iny
DA39: 158 lda catbtl,y
DA39: 159 sta addrh,x
DA39: 160 jmp mcbbyte
DA39: 161 mcbone

```

335 S.MAKECAT slinky miscellaneous routines

20-OCT-86 06:41 PAGE 1

336 S.DIAGO.SRC

### slinky miscellaneous routines

20-OCT-86 06:41 PAGE 144

144

221	dcb	\$11,\$04,\$t0f6
222	dcb	\$11,\$05,\$t0f6
223	dcb	\$11,\$06,\$t0f6
224	dcb	\$11,\$07,\$t0f6
225	dcb	\$11,\$08,\$t0f6
226	dcb	\$11,\$09,\$t0f6
227	dcb	\$11,\$0A,\$t0f6
228	dcb	\$11,\$0B,\$t0f6
229	dcb	\$11,\$0C,\$t0f6
230	dcb	\$11,\$0D,\$t0f6
231	dcb	\$11,\$0E,\$t0f6
232	dcb	zers:0,\$20,\$02
233	dcb	zers:0,\$20,\$02 ;leave pointing at VT0C
		;All done
0078	235	pascal
	0B20:	equ
236	0B20:00 00	0,0
237	0B22:06	6
238	0B23:FD 03	zers:3
239	0B25:04	4
240	0B26:52 41 4D	asc
241	0B29:AA	'RAM'
242	0B2A:FD 04	namefig
243	0B2C:FC	zers:4
244	0B2D:FD 00 00	sizefig
	0B2D:FD 00 00	zers:0,0
77	0B30:	include s.diag0.src

[illegible]



[illegible]



37	SYMBOL TABLE		SORTED BY ADDRESS		20-OCT-86 06:41 PAGE 151	
00	LOC0	? 00 BOOTBLK	? 00 PROSTAT	00 UGSPACE		
01	BADCMD	01 LOC1	01 M.MUNSE	01 PROREAD	?	
01	ISSLOT	0001 MUIDX	? 02 BOOTJMP	02 IBRDVN		
? 02	PROMLT	02 HOWMODE	? 03 PROFORM	04 IBTRK		
04	BUTMODE	04 BOPCFT	? 04 SLOT	04 M.WOXY		
? 05	ISBECT	06 GOODFR	08 M.CTL	08 M.GOOE		
0008	XH1	08 IB8UPF	0009 IQUIDX	0A ZU5D		
0C	VLAMDE	0C IECMD	00 I8STAT	0D C8CR3		
10	M.CUSOR	11 BADUNIT	0011 GUIDX	11 PCRENUM		
0013	ESUMUM	14 CTANUM	0016 NUMOPS	20 WNDLFT		
20	M.CTL2	20 MOYARN	21 BADCTL	21 WNDMTR		
22	WNDTOP	23 WNDBTM	24 CH	25 CV		
26	GRAS1	27 GRASH	? 27 TORR	28 NDRR		
28	BASL	28 BASH	2A BASZL	2B BASZB		
2C	RFLT	2C BZ	002C DSCAT	2C LNMEN		
2C	RNLT	2D BADBLK	2D RMNEN	2D VZ		
2D	RTHB	2E MASK	2E FORMAT	2F LENGTH		
30	CTOR	31 MODE	32 INVFLG	33 PROMPT		
34	YSNV	35 YSAV1	36 CSML	37 CSWH		
38	KSNL	39 KSNH	3A PCL	3B PCH		
3C	AL1	3C XQT	3C BOOTMP	3D ALB		
3E	AZL	3F AZB	40 AZL	40 M.40	?	
41	A3B	42 CMAND	42 A4L	43 UNIT		
43	PRAM	43 A4E	44 A5L	44 MACSTAT		
44	BUFER	44 PURIT	45 A5E	45 PRUFF		
45	ACC	46 BLOCK	46 XRB	47 YRB		
47	PSTAT	47 PCOUNT	47 PRLOCK	48 STATUS		
48	IOBLP	49 PADDR	49 IOBPH	49 SPNT		
4A	TEMPTR	4E RUDL	4F RUDH	4F BOOTDEV		
007B	PASCT	80 D0SER	80 M.PASCAL	8A LEED		
91	XON	93 XOFF	95 PICK	9B ESC		
AA	NAMFEG	BF CMDCUR	C1 SLTDAY	DF TERMCUR		
FC	SIZFEG	FE ZERS	FE SKPFE			
0200	IN	0200 INBUF	0214 BINL	0215 BINH		
0300	NUB1	0356 INBL	0388 SLENODE	0388 NUMBANKS		
0388	SL.SCRN1	0370 BRV	03F2 SOTEV	03F4 PREDDP		
03F5	AMPVRV	03F8 USRDR	03FB NM1	703FE IQLQC		
0400	LINEL	0438 ASAT	20438 SL.SCRN2	0478 RMSTATE		
0478	SIZETEMP	0478 MOUTMP	0478 MINL	0478 OLDC		
047D	MINX	047F MAXXL	04B8 PMTH	704B8 SL.SCRN3		
047D	EROR	04B8 MAXL	04F8 TEMP1	04FB VMODE		
04FC	ACTIABF	704FD MINYL	04FF MOUTL	0538 EXINT		
0538	SL.SCRN4	0578 MINH	0578 TEMP2	0578 XVAL		
0578	SCREEN	057C TBSR	057D MINX	057F MOUTX		
058B	SCREEN	7058B SL.SCRN5	05F8 TBSR	05F8 TEMPY		
05F8	YVAL	05F9 EXINT2	05FA ITHED	05FB ORCV		
05FC	TKWY	705FD MINH	05FE CHSRUF	05FF MOUTH		
0638	ESCHAR	70638 SL.SCRN6	0678 SL.LCSTATE	0679 OLDCUR		
067A	OLDCUR2	067B VACTV	067C TBSR	067D MAXXL		
067F	MOUTRM	068B FLACS	068B POWERUP	068B SL.SCRN7		
068F	TEMP	06FB XCOORD	706FD MAXXL	06FF TKRY		
0738	COL	70738 SL.SCRN8	0778 SL.DEVNO	07FB ORCV		
0778	MAXX	077E NUMBER	077F MOUTAT	077B NSLOT		
07F8	SL.MSLOT	07FB CURSOR	707FD MAXVH	07FF MOUTMODE		
0800	THBUF	080C BOOTBUF	1FFF DIAGSTART	2000 DIADGSET		
901E	DOSINIT	86C3 D0SSIN	B000 AMTS	BFD0 POPLAG		
B8FB	ADDRH	B7F8 SDATA	BFF9 SSTAT	BFF9 ADDRHM		
BFFA	ADDRH	BFFA SCOMD	BFFB SCHLL	BFFB DATA		



[illegible][illegible]

```

*** SUCCESSFUL ASSEMBLY := NO ERRORS
*** ASSEMBLER CREATED ON 15-JAN-84 21:28
*** TOTAL LINES ASSEMBLED 6885
*** FREE SPACE PAGE COUNT 29

```

01 PC

```

SOURCE FILE #01 =>PC
INCLUDE FILE #02 =>PC.EQUATES
INCLUDE FILE #03 =>PC.BOOTSPACE
INCLUDE FILE #04 =>PC.BOOT
INCLUDE FILE #05 =>PC.PACKET
INCLUDE FILE #06 =>PC.CREAD
INCLUDE FILE #07 =>PC.MAIN
0000: 0101 1 version equ $101 ;v1.0.1
0000: 0101 2 X6502
0000: 3 MSB ON
0000: 4 1st on,vsym,asym

```

```

0000: 7 ; PPPP RRRR 000 TTTT 000 CCC 000 L
0000: 8 ; P P R R O O T O O C C O O L
0000: 9 ; PPPP RRRR O O T O O C C O O L
0000: 10 ; P P R R O O T O O C C O O L
0000: 11 ; P P R R 000 T 000 CCC 000 LLLLL
0000: 12 ; CCC 000 N N V VEEEE RRRR TTTT EEEE RRRR
0000: 13 ; C C O O M M N V V EEE RRR R R T E R R
0000: 14 ; C C O O M M N V V EEE RRR T EEE RRR
0000: 15 ; C C O O M M N V V EEE RRR T EEE RRR
0000: 16 ; C C O O M M N V V E R R T E R R
0000: 17 ; CCC 000 N N V EEEE R R T EEEE R R

```

```

19 *****
20 *
21 * UniDisk 3.5 Driver Firmware Version 1.0.1
22 *
23 * Written by Michael Askins May 15, 1985
24 * Revised by M. Askins and R. Chiang April 10, 1986
25 *
26 * Copyright Apple Computer, Inc. 1985,1986
27 * All Rights Reserved
28 *
29 *****

```

```

0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:
0000:

```

01 PC

```

0000: 31 * Modification History:
0000: 32 *
0000: 33 *
0000: 34 *
0000: 35 * Rel Date Who Action
0000: 36 *
0000: 37 * 18 Dec 84 MSA RELEASE VERSION 0.02 (Sony)
0000: 38 * 10 Jan 85 MSA Added //c support:
0000: 39 * General conditional assembly overhead
0000: 40 * Added retries and timeouts
0000: 41 * MSslot handled correctly
0000: 42 * Finished Boot code
0000: 43 * Altered ProDOS errors - add $27 catchall
0000: 44 * Remove call to WAIT in monitor
0000: 45 * Add Boot Failure messages
0000: 46 * 22 Jan 85 MSA Add IMM reconfigure for //c version
0000: 47 * 23 Jan 85 MSA Move Comm routines to $C800 ($C900)
0000: 48 * Fixed zero page preservation
0000: 49 * 23 Jan 85 MSA RELEASE VERSION 0.03 (Apple)
0000: 50 * 25 Jan 85 MSA Swap slot dep read and boot code (//c)
0000: 51 * Add other //c differences...
0000: 52 * Add auxtype byte
0000: 53 * Fix comm error on receive packet
0000: 54 * Fix checksum to include MSBs of overhead
0000: 55 * 07 Feb 85 MSA Add COUNT support on boot fail
0000: 56 * 08 Feb 85 MSA RELEASE VERSION 1.00A (Alpha)
0000: 57 * 22 Feb 85 MSA Add bytecount in X,Y on PC calls
0000: 58 * Change hard reset time to 1 ms (was 83)
0000: 59 * Crunched code by adding CtrPhases
0000: 60 * Add zeroing of third block byte (ProDOS)
0000: 61 * 06 Mar 85 MSA fixed slot 7 goof (stack screw up)
0000: 62 * No clear phases on retries
0000: 63 * Hard reset time to 40 ms
0000: 64 * Pass #params instead of unit# and no chk
0000: 65 * Init code (all reset vs. comm reset)
0000: 66 * Add 2 bytes to pass a full 9 byte cmd
0000: 67 * Fix bytecount on retries
0000: 68 * Boot block must be $800-$01, $801<$00
0000: 69 * Remove WRREQ while waiting for motor TO
0000: 70 * Remove glitch on /DWE12 in AssignID
0000: 71 * 20 Mar 85 MSA Add interrupt on/off/poll support
0000: 72 * Reset pulse to 80 ms
0000: 73 * //c delay of 100 ms on Initial AssignID
0000: 74 * ID bytes changed
0000: 75 * Retransmit implemented (RedPack)
0000: 76 * Add send data packet retries (5)
0000: 77 * Rearrange PC stack adjust
0000: 78 * Add //c Appletalk vector
0000: 79 * 24 Mar 85 MSA Add //c millisecond wait each call
0000: 80 * 25 Mar 85 MSA RELEASE VERSION 1.00B (beta) (//e)
0000: 81 * 18 Apr 85 MSA Clear decimal mode
0000: 82 * Eight bytes are returned on stat unit#0
0000: 83 * Stat Unit#0 score<0 is rejected
0000: 84 * X and Y set to 0008 on status unit#0
0000: 85 * Enable interrupts done correctly
0000: 86 * Add unit#0 parameter count checking
0000: 87 * 22 Apr 85 MSA RELEASE VERSION 1.01B
0000: 88 * 15 May 85 MSA RELEASE VERSION 1.0

```

01 PC

```

0000: 89 * 10 Apr 86 RC removed reference to AppleTalk
0000: 90 * 10 Apr 86 RC forgot to add hi byte of auxptr in
0000: 91 * divide7 routine
0000: 92 * 10 Apr 86 RC returns write protect from old Lirons
0000: 93 * *** 10 Apr 86 RC RELEASE VERSION 1.0.1
0000: 94 * RC v1.0.1 is to be used with Tiger only
0000: 95 *
0000: 96 *

```

```

----- NEXT OBJECT FILE NAME IS PC.0
C500: 98 org $C500
C500: 99 include pc.equates

```

## 02 PC.EQUATES

## Equates

```

C500:      2 *
C500:      3 PDIDByte equ $BF
C500:      4 PCID2 equ $0
C500:      5 *
C500:      6 *****
C500:      7 *
C500:      8 * Zero Page (temps) *
C500:      9 *
C500:     10 *****
C500:     11 *
C500:     12 dssect
C500:     13 zeropage equ $0040
C500:     14 zeropage org zeropage
C500:     15 *
C500:     16 checksum dfb 0
C500:     17 topbits dfb 0
C500:     18 CMDcode dfb 0
C500:     19 CMDPcount equ *
C500:     20 CMDUnit dfb 0
C500:     21 CMDBuffer equ *
C500:     22 CMDBuffer1 dfb 0
C500:     23 CMDBufferH dfb 0
C500:     24 CMDSCode equ *
C500:     25 CMDBlock equ *
C500:     26 CMDBlock1 dfb 0
C500:     27 CMDBlockH dfb 0
C500:     28 CMDBlocks dfb 0
C500:     29 CMDSpares dfb 0
C500:     30 CMDSpares1 dfb 0
C500:     31 revbuf equ *
C500:     32 grp/ctr dfb 0
C500:     33 oddbytes dfb 0
C500:     34 statbyte equ *
C500:     35 bytecount equ *
C500:     36 bytecount1 equ *
C500:     37 next equ *
C500:     38 next1 dfb 0
C500:     39 AuxType equ *
C500:     40 bytecount equ *
C500:     41 next2 dfb 0
C500:     42 RPacketType equ *
C500:     43 next3 dfb 0
C500:     44 DeviceID equ *
C500:     45 next4 dfb 0
C500:     46 HostID equ *
C500:     47 next5 dfb 0
C500:     48 pointer equ *
C500:     49 next6 dfb 0
C500:     50 next7 dfb 0
C500:     51 buffer dw 0
C500:     52 auxptr equ *
C500:     53 buffer2 dw 0
C500:     54 slot dfb 0
C500:     55 temp equ *
C500:     56 thodd dfb 0
C500:     57 Unit dfb 0
C500:     58 WPacketType dfb 0
C500:     59 *

;ProDOS attributes byte
;This means a Jiron card

;ProDOS parameter passing area

;Current target unit

```

## 02 PC.EQUATES

## Equates

```

005C:      60 *
005C:      61 ZPSize equ *-zeropage
005C:      62 *
005C:      63 *
005C:      64 *
005C:      65 *
005C:      66 ClearIOWMs equ $CFFF
005C:      67 stack equ $100
005C:      68 *
005C:      69 *
005C:      70 *****
005C:      71 *
005C:      72 * Screenhole Storage *
005C:      73 *
005C:      74 *****
005C:      75 *
005C:      76 * The screenhole layout is as follows:
005C:      77 *
005C:      78 * //e //c
005C:      79 *
005C:      80 * ProFlag $478-n $478
005C:      81 * Retry $4F8-n $4F8
005C:      82 * SHTemp1 $578-n $578
005C:      83 * SHTempX $5F8-n $5F8
005C:      84 * SHTempY $678-n $678
005C:      85 * Power1 $6F8-n -----
005C:      86 * Power2 $778-n -----
005C:      87 * NumDevices $7F8-n $6FE
005C:      88 * SVbCL $6F8 $778
005C:      89 * SVbCH $778
005C:      90 *
005C:      91 scholes equ $473 ;Use the slot 0 sholes for temps
005C:      92 *
005C:      93 ProFlag equ scholes
005C:      94 Retry equ scholes+$80
005C:      95 SHTemp1 equ scholes+$100
005C:      96 Retry2 equ SHTemp1
005C:      97 SHTempX equ scholes+$180
005C:      98 SHTempI equ scholes+$200
005C:      99 NumDevices equ $6F9 ;Actually in slot 6
005C:     100 *
005C:     101 SVbCL equ $6F8
005C:     102 SVbCH equ $778
005C:     103 *
005C:     104 cv equ $25
005C:     105 ch equ $24
005C:     106 vtab equ $FC22
005C:     107 cout equ $DED
005C:     108 bootscrn equ $DB
005C:     109 WSlot equ $78
005C:     110 setvid equ $FE93
005C:     111 setkbd equ $FE89
005C:     112 AutoScan equ $FARA
005C:     113 Basic equ $D000
005C:     114 loc0 equ $0
005C:     115 loc1 equ $1
005C:     116 *
005C:     117 SWPROM equ $C797 ;//c bank switch to $C800

```

```

02 PC.EQUATES      Equates
C500:      C784      118 SHRT52      equ      $C784
C500:      119 *
C500:      120 *
C500:      121 * *****
C500:      122 *      General Equates      *
C500:      123 *
C500:      124 *
C500:      125 * *****
C500:      126 *
C500:      00A5 127 PRBValue      equ      $A5
C500:      00FF 128 PRBValue      equ      $FF
C500:      129 *
C500:      0000 130 PowerReset      equ      $00
C500:      0080 131 CommReset      equ      $80
C500:      132 *
C500:      0032 133 bsyTo1      equ      50
C500:      00A4 134 bsyTo2      equ      10
C500:      001E 135 statTo1      equ      30
C500:      0009 136 cmdLength      equ      9
C500:      00C3 137 packetBeg      equ      $C3
C500:      00C8 138 packetEnd      equ      $C8
C500:      0080 139 cmdmark      equ      $80
C500:      0081 140 statmark      equ      $81
C500:      0082 141 datamark      equ      $82
C500:      142 *
C500:      0007 143 iwmode      equ      $07
C500:      144 *
C500:      0000 145 SCDeviceStat      equ      0
C500:      0001 146 SCDevCDB      equ      1
C500:      0002 147 SCDevNLSat      equ      2
C500:      0003 148 SCDevInfo      equ      3
C500:      149 *
C500:      C080 150 iwm      equ      $C080
C500:      151 *
C500:      C080 152 reqClr      equ      iwm+0
C500:      C081 153 reqset      equ      iwm+1
C500:      C082 154 calcIr      equ      iwm+2
C500:      C083 155 calset      equ      iwm+3
C500:      C084 156 ca2clr      equ      iwm+4
C500:      C085 157 ca2set      equ      iwm+5
C500:      C086 158 lstrbClr      equ      iwm+6
C500:      C087 159 lstrbset      equ      iwm+7
C500:      C088 160 monClr      equ      iwm+8
C500:      C089 161 monset      equ      iwm+9
C500:      C08A 162 enable1      equ      iwm+10
C500:      C08B 163 enable2      equ      iwm+11
C500:      C08C 164 l6clr      equ      iwm+12
C500:      C08D 165 l6set      equ      iwm+13
C500:      C08E 166 l7clr      equ      iwm+14
C500:      C08F 167 l7set      equ      iwm+15
C500:      168 *
C500:      169 *
C500:      170 * Error0 codes
C500:      171 *
C500:      0001 172 noanswer      equ      1
C500:      0002 173 nomark      equ      2
C500:      0004 174 wasreset      equ      4
C500:      0008 175 bytecmp      equ      8

```

```

02 PC.EQUATES      Equates
C500:      0010 176 csmerrr      equ      $10
C500:      0020 177 nopactend      equ      $20
C500:      0040 178 bus hog      equ      $40
C500:      179 *
C500:      180 * Command Codes
C500:      181 *
C500:      0000 182 StatusCmd      equ      $00
C500:      0001 183 ReadCmd      equ      $01
C500:      0002 184 WriteCmd      equ      $02
C500:      0003 185 FormatCmd      equ      $03
C500:      0004 186 ControlCmd      equ      $04
C500:      0005 187 InitCmd      equ      $05
C500:      188 *
C500:      189 *
C500:      0040 190 Soft      equ      $01000000
C500:      191 *
C500:      0001 192 BadCmd      equ      $01
C500:      0004 193 BadCnt      equ      $04
C500:      0006 194 BusErr      equ      $06
C500:      0011 195 BadUnit      equ      $11
C500:      001F 196 Noint      equ      $1F
C500:      0021 197 BadCtl      equ      $21
C500:      0022 198 BadCtlParm      equ      $22
C500:      0027 199 IOError      equ      $27
C500:      0028 200 NoDrive      equ      $28
C500:      002B 201 WriteProt      equ      $2B
C500:      002D 202 BadBlock      equ      $2D
C500:      002F 203 Offline      equ      $2F
C500:      0068 204 LastOne      equ      Soft+NoDrive
C500:      0067 205 SoftError      equ      Soft+IOError
C500:      206 *
C500:      0010 207 SWMask1      equ      $10
C500:      208 *
C500:      0B88 209 RC1      equ      3000
C500:      0005 210 RC2      equ      5
C500:      211 *
C500:      212 *
C500:      100      include pc.boot space

```

;The soft error bit in statbyte

;Send a command pack 3000 times (3 sec)  
;Data Packs get tried only 5 times

```

C523:      2 *
C523:      3 Bootcode equ *
C523:      4 stx slot
C525:      5 *
C525:      6 ldx #SC5
C525:      7 sta Mslot
C525:      8 jsr reset
C52D:      9 *
C52D:     10 ldy #5
C52D:     11 bcl boottab,y
C52D:     12 sta endcode,y
C535:     13 dey
C536:     14 bpl bcl
C538:     15 *
C538:     16 * Now do the read from block zero
C538:     17 *
C538:     18 jsr ProDOSEntry
C538:     19 bcs bootfail ;If fail, check loc
C53D:     20 *
C53D:     21 ldx $800 ;If ($800)<1 this is no A// boot disk
C540:     22 dex
C541:     23 bne bootfail
C543:     24 *
C543:     25 ldx $801 ;If $801 is zero, no boot
C546:     26 beq bootfail
C548:     27 *
C548:     28 * It all looks okay. Jump to the code with R0 in X.
C548:     29 *
C548:     30 ldx slot
C54A:     31 asl a
C54B:     32 asl a
C54C:     33 asl a
C54D:     34 asl a
C54E:     35 tax
C54E:     36 jmp $801 ;Jump to it
C552:     37 *
C552:     38 * Do this code if the boot can't be done.
C552:     39 * If this was an autoboot (loc=$C00), continue the slot scan.
C552:     40 * If not, drop into basic after issuing appropriate message
C552:     41 *
C552:     42 bootfail ldx #>msglen-1
C554:     44 morchr $D 5F C5 bootmsg,x
C557:     45 sta bootscrn,x
C55A:     46 dex
C55B:     47 bpl morchr
C55D:     48 coma bra coma ;He's dead Jim.
C55F:     50 bootmsg asc 'Check
C570:     51 msglen equ *bootmsg
C570:     52 *
C570:     53 boottab ddb ReadCMD,$50,0,8,0,0 ;Read from 1st; blk0->$801
C576:     54 *
C576:     55 *
C576:     56 * This routine is called from the //c reset code. It forces a
C576:     57 * reset of the PC Bus. Location 0 and 1 are being used by the
C576:     58 * autostart people.

```

```

C500:      0060
C500:      3 * TheOff equ $60 ;On //c IMM in slot 6
C500:      4 * Here beginneth that code which resideth in the boot space
C500:      5 * at the time the card resteth in slot the fifth.
C500:      6 *
C500:      7 C500org equ *
C500:      8 *
C500:      9 * Auto Boot signature bytes
C500:     10 * This is also the boot (auto & PR#5) entry point.
C500:     11 *
C500:     12 ldx #920
C502:     13 ldx #500
C504:     14 ldx #803
C506:     15 *
C506:     16 cmp #0 ;Flag that this is a boot
C508:     17 bcs BootC
C50A:     18 *
C50A:     19 * Here is the ProDOS normal entry point
C50A:     20 *
C50A:     21 ProDOSEntry equ *
C50A:     22 *
C50A:     23 * Set up so that ProFLAG will have the top bit set
C50A:     24 *
C50A:     25 sec
C508:     26 bcs +3 ;Skip the clear
C508:     27 *
C508:     28 * This is the MUXface entry point
C508:     29 *
C508:     30 MUXEntry equ * ;Only use this label in //c version
C508:     31 clc
C508:     32 ldx #905
C510:     33 ror ProFLAG,x ;ProFLAG[7]=1 if ProDOS, =0 if MLI
C513:     34 clc ;This is not a boot entry
C514:     35 *
C514:     36 * Now save mslot and clear all $C800 ROMs
C514:     37 *
C514:     38 bootcases equ *
C514:     39 ldx #SC5
C516:     40 stx Mslot
C519:     41 ldx #905
C51B:     42 ldx ClearROMs ;Clear all $C800 latches but ours
C51E:     43 *
C51E:     44 jmp SWPROTO
C521:     45 BootC equ *
C521:     46 ldx #905 ;Need slot number
C521:     47 include pc.boot
C522:     101

```

04 PC:BOOT Service Boot Request 20-OCT-86 06:29 PAGE 11

```

C576: C576 60 Reset equ *
C576:A2 08 61 ldx #8
C578: C578 62 rstl equ *
C578:BD 83 C5 63 ldx rcode,x
C57B:95 02 64 sta loc0+2,x
C57D:CA C578 65 dex
C57E:10 F8 66 bpl rstl
C580:4C 02 00 67 jmp loc0+2

C583: C583 69 rcode equ *
C583:20 00 C5 70 jsr MLEntry
C586:05 71 dfb InitCMD
C587:09 00 72 dw $0009
C589:60 73 rts

C58A:01 00 75 cmlist dfb 1,0 ;One parm - the unit $00

----- NEXT OBJECT FILE NAME IS PC.1
C5F5: C5F5 103 org $C5F5
C5F5:4C 52 C5 104 jmp bootfail
C5F8:4C 76 C5 105 jmp reset
C5F8:00 106 dfb PCID2
C5FC:00 00 107 dw 0
C5FE:BF 108 dfb PDIDByte
C5FF:0A 109 dfb >ProbeEntry

----- NEXT OBJECT FILE NAME IS PC.2
C880: C880 111 org $C880
C880:4C 4C CD 112 jmp Entry
;The //c bank switch jumps here

C883: 114 include pc.packet
C883: 1 lst
C883: 2 *

```

05 PC:PACKET Send a CBUS Packet 20-OCT-86 06:29 PAGE 12

```

C883: 4 *****
C883: 5 *
C883: 6 * SendOnePacket Send a CBUS Packet
C883: 7 *
C883: 8 * This routine sends a packet of data across the
C883: 9 * bus. The protocol is as follows:
C883: 10 *
C883: 11 * REQ 12 5
C883: 12 *
C883: 13 * /BSY 1 3 4
C883: 14 *
C883: 15 * 1) Device signals ready for data
C883: 16 * 2) Host signals data imminent
C883: 17 * 3) Packet is transmitted (sync, command mark,
C883: 18 * lds, contents, checksum [nbs=1])
C883: 19 * 4) Device signals packet recieved
C883: 20 * 5) Host finishes send data cycle
C883: 21 *
C883: 22 * The bytes are sent in slow mode (32 cycles/byte)
C883: 23 * and the timing is critical. Branches which should
C883: 24 * not cross page boundaries are marked.
C883: 25 *
C883: 26 * Input: buffer (2 bytes) <- ptr to data to send
C883: 27 * bytecount (2) <- length (bytes) of data
C883: 28 * packettype (1) <- command or data packet
C883: 29 * CMUnit (1) <- # of device to receive
C883: 30 *
C883: 31 * Output: carry set- handshake error
C883: 32 * clr- bytes sent
C883: 33 *
C883: 34 *****
C883: 35 *
C883: 36 SendOnePacket equ *
C883: 37 *
C883: 38 * Prep for the transmission
C883: 39 *
C883: 40 jsr WritePrep ;Does a bunch of stuff
C883: 41 *
C883: 42 * Enable PC chain.
C883: 43 *
C883: 44 jsr enablechain ;This sets X reg
C883: 45 ldy #1mode ;This is the mode value
C883: 46 jsr Set1Nmode ;Don't mess unless we gotta
C883: 47 *
C883: 48 * Turn on the 1MW
C883: 49 *
C883: 50 lda enable2,x ;Don't disturb //c internal drive
C883: 51 lda monset,x
C883: 52 *
C883: 53 * Loop until the chain becomes unbusy
C883: 54 *
C883: 55 ldy #bsytol ;Each loop is 11 microseconds
C883: 56 wbsyl lda l7clr,x ;Test if /BSY is hi or lo
C883: 57 bmi chainunbusy ;If hl, bus is not busy
C883: 58 dey
C883: 59 bne wbsyl ;Keep trying
C883: 60 *
C883: 61 sec
C883: 62 *
C883: 63 *
C883: 64 *
C883: 65 *
C883: 66 *
C883: 67 *
C883: 68 *
C883: 69 *
C883: 70 *
C883: 71 *
C883: 72 *
C883: 73 *
C883: 74 *
C883: 75 *
C883: 76 *
C883: 77 *
C883: 78 *
C883: 79 *
C883: 80 *
C883: 81 *
C883: 82 *
C883: 83 *
C883: 84 *
C883: 85 *
C883: 86 *
C883: 87 *
C883: 88 *
C883: 89 *
C883: 90 *
C883: 91 *
C883: 92 *
C883: 93 *
C883: 94 *
C883: 95 *
C883: 96 *
C883: 97 *
C883: 98 *
C883: 99 *
C883: 100 *

```

05 PC PACKET

Send a CBus Packet

```

C89F:4C CC C9 (3) 62 * jmp sd10
C8A2: (3) 63 * Tell the bus that data is coming and send the sync bytes
C8A2: (3) 64 * Sync is groups of eight 2's separated by a 6 (mic8 cell)
C8A2: (3) 65 * (111111110011111111001111111100 ...)
C8A2: (3) 66 *
C8A2: (3) 67 *
C8A2: (3) 68 chainmbys equ *
C8A2: (3) 69 lda reqset,x ;Raise REQ
C8A5: (3) 70 *
C8A5: (3) 71 ldy #5 ;Sync plus packet begin
C8A7: (3) 72 *
C8A7: (3) 73 lda $FF ;Send out the 1st byte sync
C8A7: (3) 74 sta 17set,x
C8AC: (3) 75 *
C8AC: (3) 76 sdb lda preamble,y
C8AF: (3) 77 *
C8AF: (3) 78 *
C8AF: (3) 79 sdb asl 16clr,x ;Wait 'til buffer empty
C8B2: (3) 80 bcc sdb
C8B4: (3) 81 *
C8B4: (3) 82 sta 16set,x
C8B7: (3) 83 dey
C8B8: (3) 84 bpl sdb ;Back for more bytes
C8BA: (3) 85 *
C8BA: (3) 86 * Send over the destination ID
C8BA: (3) 87 *
C8BA: (3) 88 lda Unit
C8BA: (3) 89 ora $F80 ;Make the device ID
C8BE: (3) 90 jsr sendbyte
C8C1: (3) 91 *
C8C1: (3) 92 * Send the source ID (that's us... we're an $80)
C8C1: (3) 93 *
C8C1: (3) 94 jsr send80
C8C4: (3) 95 *
C8C4: (3) 96 * Send over the packet type (command or data)
C8C4: (3) 97 *
C8C4: (3) 98 lda Wpackettype
C8C6: (3) 99 jsr sendbyte
C8C9: (3) 100 *
C8C9: (3) 101 * Send the Auxiliary Type byte (an $80 from this rev PC)
C8C9: (3) 102 *
C8C9: (3) 103 jsr send80
C8CC: (3) 104 *
C8CC: (3) 105 * Send the status byte (null for us), and length bytes
C8CC: (3) 106 *
C8CC: (3) 107 jsr send80
C8C3: (3) 108 lda oddbytes
C8D1: (3) 109 ora $F80
C8D3: (3) 110 jsr sendbyte
C8D6: (3) 111 lda grp/ctr
C8D8: (3) 112 ora $F80
C8DA: (3) 113 jsr sendbyte
C8DD: (3) 114 *
C8DD: (3) 115 * Now send the "oddbytes" part of the packet contents
C8DD: (3) 116 *
C8DD: (3) 117 lda oddbytes ;Get # of "odd" bytes
C8DF: (3) 118 beq sob2 ;Skip if no odd bytes
C8E1: (3) 119 *

```

05 PC PACKET

Send a CBus Packet

```

C8E1:A0 FF (2) 120 ldy $FF
C8E3: (2) 121 lda tbood
C8E3: (2) 122 *
C8E3: (2) 123 sob1 asl 16clr,x ;Get the odd bytes msh's (A[7]=1)
C8E8: (2) 124 bcc sob1 ;Do a write handshake
C8EA: (2) 125 sta 16set,x
C8ED: (2) 126 iny (buffer),y ;Get the data byte
C8EE: (2) 127 ora $F80 ;Flip on the hi bit
C8F0: (2) 128 cpy oddbytes ;Are we done?
C8F2: (2) 129 bpl sob1
C8F4: (2) 130 bpl sob1
C8F6: (2) 131 *
C8F6: (2) 132 * Now send over the groups of seven contents
C8F6: (2) 133 * Currently assume there must be at least one group of 7em
C8F6: (2) 134 *
C8F6: (2) 135 sob2 equ * ;Check if there are groups to send
C8F6: (2) 136 lda grp/ctr ;=> At least one group
C8F8: (2) 137 bne sob3 ;Skip to send checksum
C8FA: (2) 138 jmp datdone
C8FD: (2) 139 *
C8FD: (2) 140 sob3 equ * ;Waste 2 cycles
C8FD: (2) 141 nop
C8FE: (2) 142 ldy #0
C8FE: (2) 143 start lda topbits
C900: (2) 144 sta 16set,x
C902: (2) 145 *
C905: (2) 146 * Send first byte
C905: (2) 147 *
C905: (2) 148 lda next1
C907: (2) 149 ora $F80
C908: (2) 150 sty temp ;Swap Y for short handshake
C90B: (2) 151 achi1 ldy 16clr,x ;Wait 'til buffer ready
C90E: (2) 152 bpl achi1
C910: (2) 153 sta 16set,x ;Send the byte
C913: (2) 154 ldy temp ;Get back Y
C915: (2) 155 *
C915: (2) 156 * Prep the next "1st" byte for next time
C915: (2) 157 *
C915: (2) 158 lda (buffer2),y
C917: (2) 159 sta next1
C919: (2) 160 asl a
C91A: (2) 161 rol topbits ;Store the top bit
C91C: (2) 162 iny ;Next byte
C91D: (2) 163 *
C91D: (2) 164 * It's possible that we're at a page boundary now. If so, bump the
C91D: (2) 165 * hi order part of the pointer.
C91D: (2) 166 *
C91D: (2) 167 bne skip1
C91F: (2) 168 inc buffer2+1
C921: (2) 169 jmp skip2
C924: (2) 170 pha
C925: (2) 171 pla
C926: (2) 172 *
C926: (2) 173 * Push us ahead by an additional 8 cycles for margin reasons
C926: (2) 174 * Plus I gotta get the topbits MSB set somehow...
C926: (2) 175 *
C926: (2) 176 skip2 equ *
C926: (2) 177 lda $400000010 ;Flip what will be MSB

```

05 PC-PACKET	Send a CBus Packet	20-OCT-86 06:29 PAGE 15	05 PC-PACKET	Send a CBus Packet	20-OCT-86 06:29 PAGE 16
C928:05 41	(3) 178 ora tophits		C971:	236 * Send the sixth byte	
C92A:85 41	(3) 179 sta tophits		C971:A5 52	(3) 238 lda next6	
C92C:	(3) 180 * Send the second byte		C973:09 80	(2) 239 ora lset,x	;Send the byte
C92C:	181 * Send the second byte		C975:9D 80 C0	(5) 240 sta lset,x	
C92C:A5 4E	(3) 183 lda next2		C978:B1 56	(5) 241 lda (buffer2),y	
C92E:09 80	(2) 184 ora lset,x		C97A:85 52	(3) 242 sta next6	
C930:9D 80 C0	(5) 185 sta lset,x		C97C:0A	(2) 243 asl a	
C933:81 56	(5) 186 lda (buffer2),y	;Send the byte	C97D:26 41	(5) 244 rol tophits	;Store the top bit
C935:85 4E	(3) 187 sta next2		C97F:C8	(2) 245 iny	;Next byte
C937:0A	(2) 188 asl a		C980:	246 * Send the last byte of the group	
C938:26 41	(5) 189 rol tophits	;Store the top bit	C980:	247 * Send the last byte of the group	
C93A:C8	(2) 190 iny	;Next byte	C980:	248 * Send the last byte of the group	
C93B:	191 * Send the third byte		C980:A5 53	(3) 249 lda next7	
C93B:	192 * Send the third byte		C982:09 80	(2) 250 ora lset,x	;Send the byte
C93B:	193 * Send the third byte		C984:9D 80 C0	(5) 251 sta lset,x	
C93B:A5 4F	(3) 194 lda next3		C987:B1 56	(5) 252 lda (buffer2),y	
C93D:09 80	(2) 195 ora lset,x		C989:85 53	(3) 253 sta next7	
C93F:9D 80 C0	(5) 196 sta lset,x	;Send the byte	C98B:0A	(2) 254 asl a	
C942:81 56	(5) 197 lda (buffer2),y		C98C:26 41	(5) 255 rol tophits	;Store the top bit
C944:85 4E	(3) 198 sta next3		C98E:C8	(2) 256 iny	;Next byte
C946:0A	(2) 199 asl a		C98F:	257 * Send the last byte of the group	
C947:26 41	(5) 200 rol tophits	;Store the top bit	C98F:	258 * Now see if we have sent enough groups of seven	
C949:C8	(2) 201 iny	;Next byte	C98F:	259 * Send the last byte of the group	
C94A:	202 * Send the fourth byte		C98F:C6 4B	(5) 260 dec grpctr	
C94A:	203 * Send the fourth byte		C991:F0 03	(5) 261 beq datdone	
C94A:	204 * Send the fourth byte		C993:	262 * Send the last byte of the group	
C94A:A5 50	(3) 205 lda next4		C993:	263 * Otherwise, back to do more. Note it's too far for a branch.	
C94C:09 80	(2) 206 ora lset,x		C993:4C 00 C9	(3) 265 jmp start	
C94E:9D 80 C0	(5) 207 sta lset,x	;Send the byte	C996:	266 * Send the last byte of the group	
C951:81 56	(5) 208 lda (buffer2),y		C996:	267 * When! Now send the damn checksum as two FN bytes	
C953:85 50	(3) 209 sta next4		C996:	268 * Send the last byte of the group	
C955:0A	(2) 210 asl a		C996:	269 datdone equ *	
C956:26 41	(5) 211 rol tophits	;Store the top bit	C996:A5 40	(3) 270 lda checksum	;c7 c6 c5 c4 c3 c2 c1 c0
C958:C8	(2) 212 iny	;Next byte	C998:09 AA	(2) 271 ora lset,x	; 1 c6 1 c4 1 c2 1 c0
C959:	213 * Send the fifth byte		C99A:8C 8C C0	(4) 272 scml ldy lset,x	;Handshake this byte
C959:	214 * After the first 256 bytes, we will cross pages here. If we did		C99A:10 FB C99A(3)	(3) 273 bpl scml	;These are even bits
C959:	215 * cross, bump the buffer pointer. If not, equalize the cases with		C99E:9D 80 C0	(5) 274 sta lset,x	
C959:	216 * seven cycles of time wasting.		C9A2:	275 * Send the last byte of the group	
C959:	217 * Send the fifth byte		C9A2:A5 40	(3) 276 lda checksum	;c7 c6 c5 c4 c3 c2 c1 c0
C959:00 85 F960(3)	(3) 218 bne skip3		C9A5:AA	(2) 277 lsr a	; 0 c7 c6 c5 c4 c3 c2 c1
C95B:86 57	(5) 219 inc buffer24		C9A5:09 AA	(2) 278 ora lset,x	; 1 c7 1 c5 1 c3 1 c1
C95D:4C 82 C9	(3) 220 jmp skip4		C9A7:20 50 CA	(6) 279 jsr sendbyte	
C960:48	(3) 221 skip3 pha		C9A8:	280 * Send the end of packet mark	
C961:68	(4) 222 pla		C9A8:	281 * Send the end of packet mark	
C962:	223 skip4 equ *		C9A8:	282 * Send the end of packet mark	
C962:	224 * Send the fifth byte		C9A8:A9 C8	(2) 283 lda lset,x	
C962:	225 * Send the fifth byte		C9AC:20 50 CA	(6) 284 jsr sendbyte	
C962:	226 * Send the fifth byte		C9AF:	285 * Wait until write underflow	
C962:A5 51	(3) 227 lda next5		C9AF:	286 * Wait until write underflow	
C964:09 80	(2) 228 ora lset,x	;Send the byte	C9AF:	287 * Wait until write underflow	
C966:9D 80 C0	(5) 229 sta lset,x		C9AF:BD 8C C0	(4) 288 sd7 lda lset,x	
C968:81 56	(5) 230 lda (buffer2),y		C9B2:29 40	(2) 289 and lset,x	
C96A:85 51	(3) 231 sta next5		C9B4:D0 F9 C9AF(3)	(3) 290 bne sd7	
C96B:85 51	(3) 232 asl a		C9B6:	291 * Send the last byte of the group	
C96D:0A	(2) 233 rol tophits	;Store the top bit	C9B6:	292 * Send the last byte of the group	
C96E:26 41	(5) 234 iny	;Next byte	C9B6:9D 80 C0	(5) 293 * Send the last byte of the group	
C970:C8	(2) 235 * Send the fifth byte		C9B7:	294 * Send the last byte of the group	

```

C9D9: 335 *
C9D9: 336 * These routines are for wasting specific amounts of time
C9D9: 337 * This code segment should not cross page boundaries.
C9D9: 338 *
C9D9: (6) 339 waste32 jsr waste14
C9D9: (2) 340 waste18 nop
C9D9: (2) 341 waste16 nop
C9D9: (2) 342 waste14 nop
C9D9: (6) 343 waste12 rts
C9D9: 344 *
C9E0: 345 *
C9E0: C9E0 346 markerr equ *
C9E0: 347 jmp dbrerror

```

```

C9B9: 294 * Now wait until the drive acknowledges receipt of the
C9B9: 295 * string or until timeout
C9B9: 296 *
C9B9: (2) 297 ldy #byte02 ;load timeout to see bsy low
C9B9: (2) 298 patch1 dey ;A little closer to an error
C9B9: (2) 299 bne sd9 ;There's still time
C9C6: (3) 300 *
C9C6: 301 * Too much time has elapsed. Drive didn't get string.
C9C6: 302 *
C9C6: (2) 303 lda #answer ;Report error in comm error byte
C9C6: (2) 304 dbrerror equ *
C9C6: C9C0 305 jsr SetM0 ;For dbrerror entry
C9C6: (6) 306 sec ;Signal a problem
C9C6: (2) 307 bcs sd10
C9C6: (3) 308 *
C9C6: 309 * See if drive has acknowledged the bytes yet
C9C6: 310 *
C9C6: (4) 311 sd9 lda l7clr,x ;Wait 'till /BSY lo
C9C6: (3) 312 bml patch1
C9C6: 313 *
C9C6: 314 * Finish the sequence
C9C6: 315 *
C9C6: (2) 316 clc ;This is a normal exit
C9C6: (4) 317 sd10 lda reqclr,x ;Set REQ lo
C9C6: (4) 318 lda l6clr,x ;Back into read mode
C9C6: 319 *
C9D2: 320 * Pull back the bytecount in all cases
C9D2: 321 *
C9D2: (6) 322 rts
C9D2: 323 *
C9D3: 324 *
C9D3: 325 * This table, when sent in reverse order, provides a
C9D3: 326 * sync pattern used to synchronize the drive IWM with
C9D3: 327 * the data stream. The first byte (last sent) is the
C9D3: 328 * packet begin mark.
C9D3: 329 *
C9D3: 330 preamble dfb packetbeg
C9D3: C9D4:FF FC F3 CF 331 synctab dfb $FF,$FC,$F3,$CF,$3F
C9D9: 332 *
C9D9: 333 *

```

```

349 C9E3: *****
350 * ReceivePack      Get a packet from bus resident
351 *
352 *
353 *
354 * REQ _____|2_____5|_____
355 *
356 * /BSY _____|1_____3_____4|_____
357 *
358 * 1) Drive signals ready to send packet
359 * 2) Host signals ready to receive data
360 * 3) Packet is transmitted (sync, mark, IDs, data,
361 *    checksum [msb=1])
362 * 4) Drive signals packet dispatched
363 * 5) Host acknowledges receipt of packet
364 *
365 * The bytes are sent in slow mode (32 cycles/byte)
366 * and the timing is critical. Branches which should
367 * not cross page boundaries are marked.
368 *
369 * Input:  buffer <- address where packet guts left
370 *
371 * Output: carry set- handshake error
372 *         clr- bytes received
373 *
374 * A <- error0 if carry set
375 * *****
376 *
377 grabstatus equ *
378 ReceivePack equ *
379 *
380 * Init the checksum
381 *
382 (2) 382 lda #500
383 (3) 383 sta checksum
384 *
385 * Copy over buffer -> buffer2
386 *
387 lda buffer
388 sta buffer2
389 lda buffer+1
390 sta buffer2+1
391 *
392 * Set up the indirect pointer for jump to 2nd part of code
393 *
394 jsr enablechain ;Set X register to $N0
395 *
396 lda l6set,x ;Prep for sense mode
397 *
398 * Now wait for BSY to go hi, signalling 'ready w/ status
399 *
400 rdhi lda l7clr,x ;Read sense
401 bpl rdhi ;Wait til a high
402 *
403 * Signal liron we're ready to receive
404 *
405 lda r6set,x ;Raise /RQ0
406 *
407 *

```

C9FD:	407 *	Wait for a byte from Liron or timeout
C9FD:	408 *	
C9FD:AO IE	(2)	ldy #statm0
C9FD:BD BC C0	(4)	ld rdn2
C9FD:10 FB	C9FF(3)	bpl l6clr,x
CA04:88	(2)	ldy rdn2
C905:30 D9	C9FD(3)	dey
CA07:	(2)	bmi marerr
CA07:	414 *	Is it the beginning of the packet?
CA07:	415 *	
CA07:C9 C3	(2)	cmp #packetbeg
CA09:D0 F4	C9FF(3)	bne rdn2
CA0B:	419 *	Okay load up the table with this stuff
CA0B:	420 *	
CA0B:	421 *	
CA0B:	422 rdn5	equ *
CA0B:	423 *	
CA0B:AO 06	(2)	ldy #6
CA0D:BD BC C0	(4)	ld rdn3
CA10:10 FB	CA0D(3)	bpl rdn3
CA14:99 48 00	(5)	sta rcbvbf,y
CA19:45 40	(2)	eor checksum
CA1B:85 40	(3)	sta checksum
CA1D:88	(2)	dey
CA1E:10 2D	CA0D(3)	bpl rdn3
CA20:	434 *	
CA20:	435 *	Set groups of seven buffer pointer buffer2
CA20:	436 *	
CA20:A5 4C	(3)	lda oddbytes
CA22:F0 27	C4AB(3)	beq start2
CA24:18	(2)	clc
CA25:65 54	(3)	adc buffer
CA27:85 56	(3)	adc buffer2
CA29:A5 55	(3)	lda buffer+1
CA2B:69 00	(2)	adc #0
CA2D:85 57	(3)	sta buffer2+1
CA2F:	445 *	
CA2F:AO 00	(2)	ldy #0
CA31:	447 *	
CA31:	448 *	Now receive the odd bytes
CA31:	449 *	
CA31:BD BC C0	(4)	start0 lda l6clr,x
CA34:10 FB	C431(3)	bpl start0
CA36:0A	(2)	asl a
CA37:85 41	(3)	sta toptbits
CA39:	454 start1	equ *
CA39:BD BC C0	(4)	lda l6clr,x
CA3C:10 FB	C439(3)	bpl start1
CA3E:06 41	(5)	asl toptbits
CA40:A0 B0 02	CA44(3)	bcs gobl
CA42:49 80	(2)	eor #80
CA44:91 54	(6)	gobl sta (buffer),y
CA46:C8	(2)	iny
CA47:CA 4C	(3)	cpy oddbytes
CA49:90 EE	C439(3)	bit start1
CA4B:	464 *	If more, branch

CA99: 523 \*  
 CA9A: (6) 524 \*  
 CA9B: 525 \*  
 CA9C: 526 \* Shift tables for use when reading. Each table should not  
 CA9D: 527 \* straddle pages.  
 CA9E: 528 \*  
 CA9F: 529 shift1 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9G: 530 dfb 0,0,0,0,0,0,0,0  
 CA9H: 531 shift2 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9I: 532 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9J: 533 shift3 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9K: 534 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9L: 535 shift4 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9M: 536 dfb \$80,\$80,\$80,\$80,\$80,\$80,\$80,\$80  
 CA9N: 537 \*  
 CA9O: 538 \*

CAB: 465 start2 equ \*  
 CAB: 466 jmp SlotDepnd  
 CAB: 467 \*  
 CAB: 468 Send80 equ \*  
 CAB: 469 lda #80  
 CAB: 470 SendByte equ \*  
 CAB: 471 ldy 16clir,x  
 CAB: 472 hpl1 SendByte  
 CAB: 473 sta 16set,x  
 CAB: 474 eor checksum  
 CAB: 475 sta checksum  
 CAB: 476  
 CAB: 477 \*  
 CAB: 478 \*  
 CAB: 479 \*  
 CAB: 480 \*  
 CAB: 481 resetchain equ \*  
 CAB: 482 jsr ClrPhases  
 CAB: 483 lda reset,x  
 CAB: 484 lda ca2set,x  
 CAB: 485 ldy #80  
 CAB: 486 jsr YMSWait  
 CAB: 487 \*  
 CAB: 488 jsr ClrPhases  
 CAB: 489 \*  
 CAB: 490 ldy #10  
 CAB: 491 \*  
 CAB: 492 YMSWait equ \*  
 CAB: 493 jsr OneMS  
 CAB: 494 dey  
 CAB: 495 bne YMSWait  
 CAB: 496  
 CAB: 497 \*  
 CAB: 498 OneMS equ \*  
 CAB: 499 ldx #200  
 CAB: 500 onems1 dex  
 CAB: 501 bne onems1  
 CAB: 502  
 CAB: 503 \*  
 CAB: 504 \*  
 CAB: 505 enablechain equ \*  
 CAB: 506 jsr SetXN0  
 CAB: 507 lda calset,x  
 CAB: 508 lda lstrbset,x  
 CAB: 509  
 CAB: 510 \*  
 CAB: 511 \*  
 CAB: 512 ClrPhases equ \*  
 CAB: 513 jsr SetXN0  
 CAB: 514 lda reqclr,x  
 CAB: 515 lda calclr,x  
 CAB: 516 lda ca2clr,x  
 CAB: 517 lda lstrbclr,x  
 CAB: 518  
 CAB: 519 \*  
 CAB: 520 \*  
 CAB: 521 SetXN0 equ \*  
 CAB: 522 ldx #560  
 CAB: 523  
 CAB: 524  
 CAB: 525  
 CAB: 526  
 CAB: 527  
 CAB: 528  
 CAB: 529  
 CAB: 530  
 CAB: 531  
 CAB: 532  
 CAB: 533  
 CAB: 534  
 CAB: 535  
 CAB: 536  
 CAB: 537  
 CAB: 538  
 CAB: 539  
 CAB: 540  
 CAB: 541  
 CAB: 542  
 CAB: 543  
 CAB: 544  
 CAB: 545  
 CAB: 546  
 CAB: 547  
 CAB: 548  
 CAB: 549  
 CAB: 550  
 CAB: 551  
 CAB: 552  
 CAB: 553  
 CAB: 554  
 CAB: 555  
 CAB: 556  
 CAB: 557  
 CAB: 558  
 CAB: 559  
 CAB: 560  
 CAB: 561  
 CAB: 562  
 CAB: 563  
 CAB: 564  
 CAB: 565  
 CAB: 566  
 CAB: 567  
 CAB: 568  
 CAB: 569  
 CAB: 570  
 CAB: 571  
 CAB: 572  
 CAB: 573  
 CAB: 574  
 CAB: 575  
 CAB: 576  
 CAB: 577  
 CAB: 578  
 CAB: 579  
 CAB: 580  
 CAB: 581  
 CAB: 582  
 CAB: 583  
 CAB: 584  
 CAB: 585  
 CAB: 586  
 CAB: 587  
 CAB: 588  
 CAB: 589  
 CAB: 590  
 CAB: 591  
 CAB: 592  
 CAB: 593  
 CAB: 594  
 CAB: 595  
 CAB: 596  
 CAB: 597  
 CAB: 598  
 CAB: 599  
 CAB: 600  
 CAB: 601  
 CAB: 602  
 CAB: 603  
 CAB: 604  
 CAB: 605  
 CAB: 606  
 CAB: 607  
 CAB: 608  
 CAB: 609  
 CAB: 610  
 CAB: 611  
 CAB: 612  
 CAB: 613  
 CAB: 614  
 CAB: 615  
 CAB: 616  
 CAB: 617  
 CAB: 618  
 CAB: 619  
 CAB: 620  
 CAB: 621  
 CAB: 622  
 CAB: 623  
 CAB: 624  
 CAB: 625  
 CAB: 626  
 CAB: 627  
 CAB: 628  
 CAB: 629  
 CAB: 630  
 CAB: 631  
 CAB: 632  
 CAB: 633  
 CAB: 634  
 CAB: 635  
 CAB: 636  
 CAB: 637  
 CAB: 638  
 CAB: 639  
 CAB: 640  
 CAB: 641  
 CAB: 642  
 CAB: 643  
 CAB: 644  
 CAB: 645  
 CAB: 646  
 CAB: 647  
 CAB: 648  
 CAB: 649  
 CAB: 650  
 CAB: 651  
 CAB: 652  
 CAB: 653  
 CAB: 654  
 CAB: 655  
 CAB: 656  
 CAB: 657  
 CAB: 658  
 CAB: 659  
 CAB: 660  
 CAB: 661  
 CAB: 662  
 CAB: 663  
 CAB: 664  
 CAB: 665  
 CAB: 666  
 CAB: 667  
 CAB: 668  
 CAB: 669  
 CAB: 670  
 CAB: 671  
 CAB: 672  
 CAB: 673  
 CAB: 674  
 CAB: 675  
 CAB: 676  
 CAB: 677  
 CAB: 678  
 CAB: 679  
 CAB: 680  
 CAB: 681  
 CAB: 682  
 CAB: 683  
 CAB: 684  
 CAB: 685  
 CAB: 686  
 CAB: 687  
 CAB: 688  
 CAB: 689  
 CAB: 690  
 CAB: 691  
 CAB: 692  
 CAB: 693  
 CAB: 694  
 CAB: 695  
 CAB: 696  
 CAB: 697  
 CAB: 698  
 CAB: 699  
 CAB: 700  
 CAB: 701  
 CAB: 702  
 CAB: 703  
 CAB: 704  
 CAB: 705  
 CAB: 706  
 CAB: 707  
 CAB: 708  
 CAB: 709  
 CAB: 710  
 CAB: 711  
 CAB: 712  
 CAB: 713  
 CAB: 714  
 CAB: 715  
 CAB: 716  
 CAB: 717  
 CAB: 718  
 CAB: 719  
 CAB: 720  
 CAB: 721  
 CAB: 722  
 CAB: 723  
 CAB: 724  
 CAB: 725  
 CAB: 726  
 CAB: 727  
 CAB: 728  
 CAB: 729  
 CAB: 730  
 CAB: 731  
 CAB: 732  
 CAB: 733  
 CAB: 734  
 CAB: 735  
 CAB: 736  
 CAB: 737  
 CAB: 738  
 CAB: 739  
 CAB: 740  
 CAB: 741  
 CAB: 742  
 CAB: 743  
 CAB: 744  
 CAB: 745  
 CAB: 746  
 CAB: 747  
 CAB: 748  
 CAB: 749  
 CAB: 750  
 CAB: 751  
 CAB: 752  
 CAB: 753  
 CAB: 754  
 CAB: 755  
 CAB: 756  
 CAB: 757  
 CAB: 758  
 CAB: 759  
 CAB: 760  
 CAB: 761  
 CAB: 762  
 CAB: 763  
 CAB: 764  
 CAB: 765  
 CAB: 766  
 CAB: 767  
 CAB: 768  
 CAB: 769  
 CAB: 770  
 CAB: 771  
 CAB: 772  
 CAB: 773  
 CAB: 774  
 CAB: 775  
 CAB: 776  
 CAB: 777  
 CAB: 778  
 CAB: 779  
 CAB: 780  
 CAB: 781  
 CAB: 782  
 CAB: 783  
 CAB: 784  
 CAB: 785  
 CAB: 786  
 CAB: 787  
 CAB: 788  
 CAB: 789  
 CAB: 790  
 CAB: 791  
 CAB: 792  
 CAB: 793  
 CAB: 794  
 CAB: 795  
 CAB: 796  
 CAB: 797  
 CAB: 798  
 CAB: 799  
 CAB: 800  
 CAB: 801  
 CAB: 802  
 CAB: 803  
 CAB: 804  
 CAB: 805  
 CAB: 806  
 CAB: 807  
 CAB: 808  
 CAB: 809  
 CAB: 810  
 CAB: 811  
 CAB: 812  
 CAB: 813  
 CAB: 814  
 CAB: 815  
 CAB: 816  
 CAB: 817  
 CAB: 818  
 CAB: 819  
 CAB: 820  
 CAB: 821  
 CAB: 822  
 CAB: 823  
 CAB: 824  
 CAB: 825  
 CAB: 826  
 CAB: 827  
 CAB: 828  
 CAB: 829  
 CAB: 830  
 CAB: 831  
 CAB: 832  
 CAB: 833  
 CAB: 834  
 CAB: 835  
 CAB: 836  
 CAB: 837  
 CAB: 838  
 CAB: 839  
 CAB: 840  
 CAB: 841  
 CAB: 842  
 CAB: 843  
 CAB: 844  
 CAB: 845  
 CAB: 846  
 CAB: 847  
 CAB: 848  
 CAB: 849  
 CAB: 850  
 CAB: 851  
 CAB: 852  
 CAB: 853  
 CAB: 854  
 CAB: 855  
 CAB: 856  
 CAB: 857  
 CAB: 858  
 CAB: 859  
 CAB: 860  
 CAB: 861  
 CAB: 862  
 CAB: 863  
 CAB: 864  
 CAB: 865  
 CAB: 866  
 CAB: 867  
 CAB: 868  
 CAB: 869  
 CAB: 870  
 CAB: 871  
 CAB: 872  
 CAB: 873  
 CAB: 874  
 CAB: 875  
 CAB: 876  
 CAB: 877  
 CAB: 878  
 CAB: 879  
 CAB: 880  
 CAB: 881  
 CAB: 882  
 CAB: 883  
 CAB: 884  
 CAB: 885  
 CAB: 886  
 CAB: 887  
 CAB: 888  
 CAB: 889  
 CAB: 890  
 CAB: 891  
 CAB: 892  
 CAB: 893  
 CAB: 894  
 CAB: 895  
 CAB: 896  
 CAB: 897  
 CAB: 898  
 CAB: 899  
 CAB: 900  
 CAB: 901  
 CAB: 902  
 CAB: 903  
 CAB: 904  
 CAB: 905  
 CAB: 906  
 CAB: 907  
 CAB: 908  
 CAB: 909  
 CAB: 910  
 CAB: 911  
 CAB: 912  
 CAB: 913  
 CAB: 914  
 CAB: 915  
 CAB: 916  
 CAB: 917  
 CAB: 918  
 CAB: 919  
 CAB: 920  
 CAB: 921  
 CAB: 922  
 CAB: 923  
 CAB: 924  
 CAB: 925  
 CAB: 926  
 CAB: 927  
 CAB: 928  
 CAB: 929  
 CAB: 930  
 CAB: 931  
 CAB: 932  
 CAB: 933  
 CAB: 934  
 CAB: 935  
 CAB: 936  
 CAB: 937  
 CAB: 938  
 CAB: 939  
 CAB: 940  
 CAB: 941  
 CAB: 942  
 CAB: 943  
 CAB: 944  
 CAB: 945  
 CAB: 946  
 CAB: 947  
 CAB: 948  
 CAB: 949  
 CAB: 950  
 CAB: 951  
 CAB: 952  
 CAB: 953  
 CAB: 954  
 CAB: 955  
 CAB: 956  
 CAB: 957  
 CAB: 958  
 CAB: 959  
 CAB: 960  
 CAB: 961  
 CAB: 962  
 CAB: 963  
 CAB: 964  
 CAB: 965  
 CAB: 966  
 CAB: 967  
 CAB: 968  
 CAB: 969  
 CAB: 970  
 CAB: 971  
 CAB: 972  
 CAB: 973  
 CAB: 974  
 CAB: 975  
 CAB: 976  
 CAB: 977  
 CAB: 978  
 CAB: 979  
 CAB: 980  
 CAB: 981  
 CAB: 982  
 CAB: 983  
 CAB: 984  
 CAB: 985  
 CAB: 986  
 CAB: 987  
 CAB: 988  
 CAB: 989  
 CAB: 990  
 CAB: 991  
 CAB: 992  
 CAB: 993  
 CAB: 994  
 CAB: 995  
 CAB: 996  
 CAB: 997  
 CAB: 998  
 CAB: 999  
 CAB: 1000

20-OCT-86 06:29 PAGE 23

05 PC-PACKET Receive a CBus Packet

```

CADA:          540 *
CADA:          541 SendData equ *
CADA:A9 05      (2) 542 lda #>RC2
CADC:A0 00      (2) 543 ldy #<RC2
CADC:20 FD CA   (6) 544 jsr SendPile
CCE1:90 05      CBE8 (3) 545 bcc sbount
CCE1:A9 80      (2) 546 lda #ComReset
CCE1:20 98 CF   (6) 547 jsr AssignID
CCE8:          CBE8 (6) 548 sbount equ *
CCE8:60         (6) 549 rts
CCE9:          550 *
CCE9:          551 *
CCE9:          CBE9 552 SendPack equ *
CCE9:20 FD CA   (6) 553 jsr SendPile
CCEC:90 FA      CBE8 (3) 554 bcc sbount
CCEB:A9 80      (2) 555 lda #ComReset
CAR0:20 98 CF   (6) 556 jsr AssignID
CAR3:          557 *
CAR3:AD F8 06   (4) 558 lda SVbCl
CAR6:85 4D      (3) 559 sta bytecount1
CAR8:AD 78 07   (4) 560 lda SVbCl
CARB:85 4E      (3) 561 sta bytecounth
CARD:          562 *
CARD:          CAFE 563 SendPile equ *
CARD:A9 B8      (2) 564 lda #>RC1
CARF:A0 0B      (2) 565 ldy #<RC1
CB01:          566 *
CB01:          CB01 567 AltSendPile equ *
CB01:A6 58      (3) 568 ldx slot
CB03:90 F3 04   (5) 569 sta Retry,x
CB06:98         (2) 570 tya
CB07:90 73 05   (5) 571 sta Retry2,x
CB0A:          572 *
CB0A:          573 * SendPack destroys the bytecount
CB0A:          574 *
CB0A:          CB0A 575 spilel equ *
CB0A:A5 4D      (3) 576 lda bytecount1
CB0C:8D F8 06   (4) 577 sta SVbCl
CB0F:A5 4E      (3) 578 lda bytecounth
CB11:8D 78 07   (4) 579 sta SVbCl
CB14:          580 *
CB14:          581 jsr SendOnePack
CB17:          582 *
CB17:AD F8 06   (4) 583 lda SVbCl
CB1A:85 4D      (3) 584 sta bytecount1
CB1C:AD 78 07   (4) 585 lda SVbCl
CB1F:85 4E      (3) 586 sta bytecounth
CB21:          587 *
CB21:90 0C      CBEF (3) 588 bcc spilout
CB23:A6 58      (3) 589 ldx slot
CB25:DE F3 04   (7) 590 dec Retry,x
CB28:D0 E0      CB0A (3) 591 bne spilel
CB2A:DE 73 05   (7) 592 dec Retry2,x
CB2D:10 DB      CB0A (3) 593 bpl spilel
CB2F:60         (6) 594 spilout rts
CB30:          595 *
CB30:          CB30 596 RecPack equ *
CB30:A4 58      (3) 597 ldy slot

```

20-OCT-86 06:29 PAGE 24

05 PC-PACKET Receive a CBus Packet

```

CB32:A9 05      (2) 598 lda #>RC2
CB34:99 F3 04   (5) 599 sta Retry,y
CB37:          CB37 600 rpk1 equ *
CB37:20 F3 C9   (6) 601 jsr ReceivePack
CB3A:90 0F      CB4B (3) 602 bcc rpoint
CB3C:A0 01      (2) 603 ldy #1
CB3E:20 70 CA   (6) 604 jsr YMSWait
CB41:20 C0 C9   (6) 605 jsr derror
CB44:A6 58      (3) 606 ldx slot
CB46:DE F3 04   (7) 607 dec Retry,x
CB49:D0 EC      CB37 (3) 608 bne rpk1
CB4B:60         (6) 610 rts
CB4C:          611 *
CB4C:          612 *

```

;Recycle handshake and set carry

;Carry set still

Divide by 7 routine

```

CB4C: 614 *****
CB4C: 615 * Divide? Do DIV and MOD 7 and set auxptr *
CB4C: 616 *
CB4C: 617 * This routine divides the bytcount by seven. The
CB4C: 618 * quotient gives the number of groups of seven bytes to
CB4C: 619 * be sent, and the remainder gives the number of "odd"
CB4C: 620 * bytes.
CB4C: 621 *
CB4C: 622 * Input: bytcount,l,h <- # of bytes to write
CB4C: 623 * buffer <- pointer to data
CB4C: 624 * Output: auxptr <- pointer to speed up csumming
CB4C: 625 * oddbytes <- bytcount MOD 7
CB4C: 626 * grp/ctr <- bytcount DIV 7
CB4C: 627 *
CB4C: 628 *
CB4C: 629 *****
CB4C: 630 *
CB4C: 631 pdiv7tab dfb 0,36,73
CB4C: 632 pmod7tab dfb 0,4,1
CB4C: 633 div7tab dfb 0,1,2,4,9,18
CB4C: 634 mod7tab dfb 0,1,2,4,1,2
CB4C: 635 *
CB4C: 636 auxptrinc dfb 0,57F,57F
CB4C: 637 *
CB4C: 638 WriteRep equ *
CB4C: 639 Divide7 equ *
CB4C: 640 *
CB4C: 641 * Set up auxptr <- buffer+$80 if $0FF < bytcount < $200
CB4C: 642 * or auxptr <- buffer+$100 if $1FF < bytcount
CB4C: 643 *
CB4C: 644 ldx bytcountlh ;0, 1 or 2
CB4C: 645 beq noauxptr ;Auxptr used only for full pages
CB4C: 646 *
CB4C: 647 ldx buffer+1
CB4C: 648 sta auxptr+1 ;Copy over hi order part
CB4C: 649 *
CB4C: 650 ldx $80 ;Anticipate smaller bytcount
CB4C: 651 cpx #1 ;Check bytcount
CB4C: 652 beq sap1 ;=> $0FF < bytcount < $200
CB4C: 653 *
CB4C: 654 inc auxptr+1 ;Add $100 to bytcount instead
CB4C: 655 ldx #0 ;Make sure lo order unaltered
CB4C: 656 cbc auxptr+1
CB4C: 657 ldx buffer
CB4C: 658 sta auxptr
CB4C: 659 bcc noauxptr ;skip if no carry
CB4C: 660 inc auxptr+1 ;don't forget me
CB4C: 661 *
CB4C: 662 * Now look up the first order guess for DIV and MOD. X still has
CB4C: 663 * bytcount DIV 256.
CB4C: 664 *
CB4C: 665 noauxptr equ *
CB4C: 666 ldx pdiv7tab,x
CB4C: 667 sta grp/ctr
CB4C: 668 ldx pmod7tab,x
CB4C: 669 sta oddbytes
CB4C: 670 *
CB4C: 671 * Now add in the mods and divs for each of the five hi order

```

Divide by 7 routine

```

CB86: 672 * bits in the lo order bytcount, correcting each time MOD becomes
CB86: 673 * bigger than 6.
CB86: 674 *
CB86:A2 05 (2) 675 ldx #5 ;Do for five bits
CB88:A5 4D (3) 676 ldx bytcountl ;Store lo order for shifting
CB8A:85 59 (3) 677 sta temp ;Save lo three for later
CB8C:29 07 (2) 678 and #40000111
CB8E:A8 (2) 679 tay
CB8F: 680 *
CB8F: 681 divide3 equ *
CB8F: 682 shl temp ;C <- next from bytcountl
CB8F: 683 bcc divide2 ;If clear, no effect on DIV,MOD
CB91:9D 3B (4) 684 ldx mod7tab,x ;Get MOD for 2^n
CB93:BD 38 CB (4) 685 divide4 equ *
CB96: (2) 686 clc
CB96:18 (2) 687 adc oddbytes ;Got new MOD value
CB97:65 4C (3) 688 cmp #7 ;Is it too big?
CB98:C9 07 (2) 689 blt divide1 ;=> NO leave MOD - 0->C
CB99:90 02 CB9F(3) 690 sbc #7 ;Bring MOD under 7 - C still set
CB9F: CB9F 691 divide1 equ *
CB9F: 692 sta oddbytes ;Get DIV for this 2^n
CB9F: 693 ldx div7tab,x ;Add to DIV along with correction (C)
CB9F: 694 adc grp/ctr ;Update the DIV
CB9F: 695 sta grp/ctr ;One less bit to deal with
CB9F: 696 divide2 equ *
CB9F: 697 dex divide5 ;Escape after 6 times through loop
CB9F: 698 bne divide3 ;Take brnch 1st 5 loops
CB9F: 699 tya ;Get back the last three bits
CB9F: 700 *
CB9F: 701 jmp divide4 ;Sixth pass add in remains
CB9F: 702 *
CB9F: 703 *
CB9F: 704 divide5 equ *
CB9F: 705 *
CB9F: 706 *

```

05 PC PACKET	Checksum Prepass	20-OCT-86 06:29 PAGE 27	05 PC PACKET	Get topbits byte for odds	20-OCT-86 06:29 PAGE 28
CBB1:	708 *****		CBE2:	766 *****	
CBB1:	709 *	Precheck	CBE2:	767 *	
CBB1:	710 *	Does the checksumming prepass *	CBE2:	768 * DetTopBits	Get topbits for odd bytes *
CBB1:	711 *		CBE2:	769 *	
CBB1:	712 *	bytecount <- bytes in buffer	CBE2:	770 *	Also sets buffer2 pointer to pointer at groups of
CBB1:	713 *	buffer <- pointer to data to send	CBE2:	771 *	seven bytes.
CBB1:	714 *	auxptr <- extra pointer to speed process	CBE2:	772 *	
CBB1:	715 *	checksum <- 8 bit XOR of data to be sent	CBE2:	773 *	Input: oddbytes <- # of "odd" bytes
CBB1:	716 *		CBE2:	774 *	buffer <- pointer to data
CBB1:	717 *****		CBE2:	775 *	tboodd <- topbits for odd bytes
CBB1:	718 *		CBE2:	776 *	buffer2 <- buffer+oddbytes
CBB1:	719 PreCheck equ *		CBE2:	777 *	
CBB1:	720 *	Checksum any full pages	CBE2:	778 *****	
CBB1:	721 *		CBE2:	779 *	
CBB1:	722 *	lda buffer+1	CBE2:	780 DetTopBits equ *	
CBB1:	723	pha	CBE2:	781 *	
CBB1:	724	lda #0	CBE2:	782	ldy oddbytes
CBB1:	725	lda #0	CBE2:	783	dey
CBB1:	726	bytecount	CBE2:	784	lda #0
CBB1:	727	beq lastpass	CBE2:	785	sta tboodd
CBB1:	728 xor2	equ *	CBE2:	786 *	
CBB1:	729	ldy auxptrinc,x	CBE2:	787 gtbob	lda (buffer),y
CBB1:	730 xor1	equ *	CBE2:	788	asl a
CBB1:	731	eor (buffer),y	CBE2:	789	ror tboodd
CBB1:	732	eor (auxptr),y	CBE2:	790	dey
CBB1:	733	dey	CBE2:	791	bp1 gtbob
CBB1:	734	bne xor1	CBE2:	792	sec
CBB1:	735	eor (buffer),y	CBE2:	793	ror tboodd
CBB1:	736	eor (auxptr),y	CBE2:	794 *	
CBB1:	737 *	Have to deal with 0 case	CBE2:	795	lda oddbytes
CBB1:	738 *	Now move the buffer up for next section	CBE2:	796	cic
CBB1:	739 *		CBE2:	797	adc buffer
CBB1:	740	cpx #1	CBE2:	798	sta buffer2
CBB1:	741	beq xor5	CBE2:	799	lda buffer+1
CBB1:	742	inc buffer+1	CBE2:	800	adc #0
CBB1:	743 xor5	inc buffer+1	CBE2:	801	sta buffer2+1
CBB1:	744 *		CBE2:	802 *	
CBB1:	745 lastpass equ *		CBE2:	803 *	
CBB1:	746 *	Do the remaining less than a page with a single pointer	CBE2:		
CBB1:	747 *		CBE2:		
CBB1:	748 *		CBE2:		
CBB1:	749	ldy bytecount	CBE2:		
CBB1:	750	beq xor4	CBE2:		
CBB1:	751	eor (buffer),y	CBE2:		
CBB1:	752 xor3	eor (buffer),y	CBE2:		
CBB1:	753	dey	CBE2:		
CBB1:	754	bne xor3	CBE2:		
CBB1:	755	eor (buffer),y	CBE2:		
CBB1:	756 *	Last damn (0th) byte	CBE2:		
CBB1:	757 *	Store result away. Retrieve old buffer value.	CBE2:		
CBB1:	758 *		CBE2:		
CBB1:	759 xor4	equ *	CBE2:		
CBB1:	760	sta checksum	CBE2:		
CBB1:	761	pla	CBE2:		
CBB1:	762	sta buffer+1	CBE2:		
CBB1:	763 *		CBE2:		
CBB1:	764 *		CBE2:		

```

CC01: *****
CC01: 805 *
CC01: 806 *
CC01: 807 * Sun          Set up next buffer and tophits *
CC01: 808 *
CC01: 809 * Primes the pipe for the group of seven bytes routine *
CC01: 810 * setting the tophits byte and the "next" buffer. *
CC01: 811 * The routine also advances the buffer pointer by 7 to *
CC01: 812 * prepare for the groups of seven transfer. *
CC01: 813 *
CC01: 814 * Input:  buffer2  <- points to groups of 7 data *
CC01: 815 * Output: next1,7  <- first 7 bytes in buffer *
CC01: 816 * tophits  <- #38s of first 7 bytes *
CC01: 817 *
CC01: 818 *****
CC01: 819 *
CC01: 820 Sun equ *
CC01: 821 *
CC01: 822 * Copy first seven bytes into the pipeline *
CC01: 823 *
CC01: 824 *
CC01: 825 *
CC01: 826 *
CC01: 827 *
CC01: 828 *
CC01: 829 *
CC01: 830 *
CC01: 831 *
CC01: 832 *
CC01: 833 *
CC01: 834 *
CC01: 835 *
CC01: 836 *
CC01: 837 *
CC01: 838 *
CC01: 839 *
CC01: 840 *
CC01: 841 *
CC01: 842 *
CC01: 843 *
CC01: 844 *
CC01: 845 *
CC01: 846 *
CC01: 847 *

```

```

CC20: 849 *
CC20: 850 * X is slot16, Y is the desired mode
CC20: 851 *
CC20: 852 * Set up the IMW mode register. Extreme care should be taken
CC20: 853 * here. Setting the mode byte with indexed stores causes a
CC20: 854 * false byte to be written a cycle before the real value is
CC20: 855 * written. This false value, if it enables the timer, causes
CC20: 856 * the IMW Rev A to pop the motor on, inhibiting the setting
CC20: 857 * of the mode until the motor times out! We avoid this by
CC20: 858 * setting the mode byte only when it is not what we want, and if
CC20: 859 * it's not we stay here until we see that it is what we want.
CC20: 860 *
CC20: 861 SetIMWmode equ *
CC20: 862 *
CC20: 863 *
CC20: 864 *
CC20: 865 *
CC20: 866 *
CC20: 867 *
CC20: 868 *
CC20: 869 *
CC20: 870 *
CC20: 871 *
CC20: 872 *
CC20: 873 *
CC20: 874 *
CC20: 875 *
CC20: 876 *
CC20: 877 *
CC20: 878 *
CC20: 879 *
CC20: 880 *
CC20: 881 *
CC20: 882 *
CC20: 883 *
CC20: 884 *
CC20: 885 *
CC20: 886 *
CC20: 887 *
CC20: 888 *
CC20: 889 *
CC20: 890 *
CC20: 891 *
CC20: 892 *
CC20: 893 *
CC20: 894 *
CC20: 895 *
CC20: 896 *
CC20: 897 *
CC20: 898 *
CC20: 899 *
CC20: 900 *
CC20: 901 *
CC20: 902 *
CC20: 903 *
CC20: 904 *
CC20: 905 *
CC20: 906 *

```

```

05 PC.PACKET      Set the INM mode reg      20-OCT-86 06:29 PAGE 31
CC56:86 4B      (3) 907      stx      grp7ctr
CC58:A2 03      (2) 908      ldx      #3
CC5A:0A      (2) 909      times7      a
CC5B:26 4B      (2) 910      rol      grp7ctr
CC5D:CA      (2) 911      dex
CC5E:D0 FA      CC5A(3) 912      bne      times7
CC60:18      (2) 913      clc
CC61:65 4C      (3) 914      adc      oddbytes
CC63:90 02      CC67(3) 915      bcc      t71
CC65:E6 4B      (5) 916      inc      grp7ctr
CC67:84 4C      (3) 917      sty      oddbytes
CC69:38      (2) 918      sec
CC6A:E5 4C      (3) 919      sbc      oddbytes
CC6C:B0 02      CC70(3) 920      bcs      t72
CC6E:C6 4B      (5) 921      dec      grp7ctr
CC70:A4 4B      (3) 922      ldy      grp7ctr
CC72:60      (6) 923      rts
CC73:      924 *
CC73:      925 *
CC73:      115      include pc.cread
CC73:      1      slotDeplad equ *
CC73:      2      start25 equ *
CC73:      3      ldy #0
CC73:A5 4B      (3) 4      ldx      grp7ctr
CC77:48      (3) 5      pha
CC78:D0 03      CC7D(3) 6      bne      start35
CC7A:4C 0A CD      7      jmp      done5
CC7D:      8 *
CC7D:      9 * Okay, get the groups of seven
CC7D:      10 * Start by getting the topbits for this group of seven
CC7D:      11 *
CC7D:      12      start35 equ *
CC7D:      13      ldx      16clr*TheOff
CC7D:      14      bpl      start35
CC80:10 FB      CC7D(3) 15      sta      temp
CC82:85 59      (3) 16 *
CC84:      17 * Split up the seven bits into two indices for topbit tables
CC84:      18 *
CC84:      19      lsr      a ;0 1 d1 d2 d3 d4 d5 d6
CC85:4A      (2) 20      lsr      a ;0 0 1 d1 d2 d3 d4 d5
CC86:4A      (2) 21      lsr      a ;0 0 0 1 d1 d2 d3 d4
CC87:29 0F      (2) 22      and      #400001111
CC88:A5 59      (2) 23      tax
CC89:AA      (2) 24      ldx      temp ;First index into the tables
CC8C:29 07      (2) 25      and      #400000111
CC8E:85 59      (3) 26      sta      temp ;Keep for last three bytes
CC90:      27 *
CC90:      28 * Read the 1st byte, reinit its msb, store and checksum it
CC90:      29 *
CC90:      30      ldx      16clr*TheOff
CC93:10 FB      CC90(3) 31      bpl      +3
CC95:5D 9A CA      (4) 32      eor      shift1,x
CC96:91 56      (6) 33      sta      (buffer2),y
CC9A:45 40      (3) 34      eor      checksum
CC9C:85 40      (3) 35      sta      checksum
CC9E:C8      (2) 36      iny
CC9F:      37 *
CC9F:      38 * Now, the second Y turn over occurs at this point in the

```

```

06 PC.CREAD      Set the INM mode reg      20-OCT-86 06:29 PAGE 32
CC9F:      39 * loop. Update the buffer pointer if it occurred.
CC9F:      40 *
CC9F:D0 02      CC93(3) 41      bne      +4
CCAF:E6 57      (5) 42      inc      buffer2+1
CCB3:      43 *
CCB3:      44 * Now the second byte
CCB3:      45 *
CCB3:AD EC C0      (4) 46      ldx      16clr*TheOff
CCB6:10 FB      CCB3(3) 47      bpl      +3
CCB8:5D AA CA      (4) 48      eor      shift2,x
CCBA:91 56      (6) 49      sta      (buffer2),y
CCBD:45 40      (3) 50      eor      checksum
CCBF:85 40      (3) 51      sta      checksum
CCB1:C8      (2) 52      iny
CCB2:      53 *
CCB2:      54 * Now the third byte
CCB2:      55 *
CCB2:AD EC C0      (4) 56      ldx      16clr*TheOff
CCB5:10 FB      CCB2(3) 57      bpl      +3
CCB7:5D AA CA      (4) 58      eor      shift3,x
CCBA:91 56      (6) 59      sta      (buffer2),y
CCBC:45 40      (3) 60      eor      checksum
CCBE:85 40      (3) 61      sta      checksum
CCC0:C8      (2) 62      iny
CCC1:      63 *
CCC1:      64 * Now the fourth byte
CCC1:      65 *
CCC1:AD EC C0      (4) 66      ldx      16clr*TheOff
CCC4:10 FB      CCC1(3) 67      bpl      +3
CCC6:5D AA CA      (4) 68      eor      shift4,x
CCC9:91 56      (6) 69      sta      (buffer2),y
CCCB:45 40      (3) 70      eor      checksum
CCCD:85 40      (3) 71      sta      checksum
CCCF:C8      (2) 72      iny
CCD0:      73 *
CCD0:      74 * The first Y turn over occurs at this point in the loop. Update
CCD0:      75 * the buffer pointer if it occurred.
CCD0:      76 *
CCD0:D0 02      CCD4(3) 77      bne      +4
CCD2:E6 57      (5) 78      inc      buffer2+1
CCD4:      79 *
CCD4:A6 59      (3) 80      ldx      temp ;Now we need the other index
CCD6:      81 *
CCD6:      82 * Now the fifth byte
CCD6:      83 *
CCD6:AD EC C0      (4) 84      ldx      16clr*TheOff
CCD9:10 FB      CCD6(3) 85      bpl      +3
CCDB:5D AA CA      (4) 86      eor      shift2,x
CCDE:91 56      (6) 87      sta      (buffer2),y
CCD0:45 40      (3) 88      eor      checksum
CCD2:85 40      (3) 89      sta      checksum
CCD4:C8      (2) 90      iny
CCD5:      91 *
CCD5:      92 * Now the sixth byte
CCD5:      93 *
CCD5:AD EC C0      (4) 94      ldx      16clr*TheOff
CCD8:10 FB      CCD5(3) 95      bpl      +3
CCDA:5D AA CA      (4) 96      eor      shift3,x

```

```

C02D:91 56      (6) 97 sta (buffer2),y ;Store it away
C03F:45 40      (3) 98 eor checksum ;Add it to the checksum
C0F1:85 40      (3) 99 sta checksum
C0F3:C8         (2) 100 iny
C0F4:          (2) 101 *
C0F4:          102 * And, finally, the seventh byte
C0F4:          103 *
C0F4:          104 lda 16clr+TheOff
C0F7:10 FB CCF4(3) 105 bpl *3 ;Back 1 instruction
C0F9:5D CA CA (4) 106 eor shift4,x ;Recombine the MSB with data
C0FC:91 56      (6) 107 sta (buffer2),y ;Store it away
C0FE:45 40      (3) 108 eor checksum ;Add it to the checksum
C000:85 40      (3) 109 sta checksum
C002:C8         (2) 110 iny
C003:          (2) 111 *
C003:          112 * Now see if this is the last group of seven to receive
C003:          113 *
C003:          114 dec grp7ctr
C003:C6 4B      (5) 115 beq done5 ;Go to get the checksum etc
C005:F0 03 C00A(3) 116 jmp start35 ;Another tobits ...
C007:4C 7D CC (3) 117 *
C00A:          118 * Get and reconstruct the checksum
C00A:          119 *
C00A:          120 done5 equ *
C00A:AD EC C0 (4) 121 lda 16clr+TheOff
C00D:10 FB C00A(3) 122 bpl *3
C00F:85 59      (3) 123 sta temp ;1 c6 1 c4 1 c2 1 c0
C011:          (2) 124 *
C011:68         (4) 125 pla ;Restore groups of 7 counter
C012:85 4B      (3) 126 sta
C014:AD EC C0 (4) 127 lda 16clr+TheOff ;1 c1 1 c5 1 c3 1 c1
C017:10 FB C014(3) 128 bpl *3
C019:38         (2) 129 sec
C01A:2A         (2) 130 rol a
C01B:25 59      (3) 131 and temp ;c7 c6 c5 c4 c3 c2 c1 c0
C01D:45 40      (3) 132 eor checksum ;When we're done, should be zero
C01F:          (3) 133 *
C01F:          134 * Get the packet end mark. Is it correct?
C01F:          135 *
C01F:AC EC C0 (4) 136 rdba5 ldy 16clr+TheOff ;Preserve A
C022:10 FB C01F(3) 137 bpl rdba5
C024:          (3) 138 *
C024:C0 C8      (2) 139 cpy #packetend
C026:D0 1C C041(3) 140 bne npenderr5
C028:          (2) 141 *
C028:          142 * Didn't have time before to checksum oddbytes. Do it now
C028:          143 * A still has the partial checksum
C028:          144 *
C028:A6 4C      (3) 145 ldx oddbytes
C02A:F0 08 C034(3) 146 beq icht15
C02C:A0 00      (2) 147 ldy #0
C02E:51 54      (5) 148 icht5 eor (buffer),y
C030:C8         (2) 149 iny
C031:CA         (2) 150 dex
C032:D0 FA C02E(3) 151 bne icht5
C034:          (2) 152 *
C034:          153 * Okay, checksum oughta be zero. If not, checksum error.
C034:          154 *

```

```

C034:          (2) 155 icht15 equ *
C034:AA         (2) 156 tax
C035:D0 11 C048(3) 157 bne cerror5
C037:          (2) 158 *
C037:          159 * Wait for /BSY to go low
C037:          160 *
C037:          C037 161 lsbyswait5 equ *
C037:AD ED C0 (4) 162 lda 16set+TheOff
C03A:AD EC C0 (4) 163 rdb45 lda 17clr+TheOff
C03D:30 FB C03A(3) 164 bml rdb45
C03F:          165 *
C03F:          166 * Got the bytes, now acknowledge their receipt
C03F:          167 *
C03F:AD ED C0 (4) 168 lda reqclr+TheOff ;Lower REQ
C042:          (2) 170 clc
C043:60         (6) 171 rts
C044:          (2) 172 *
C044:A9 20      (2) 173 npenderr5 lda #nopactend
C046:D0 02 C04A(3) 174 bne gerror5
C048:A9 10      (2) 175 cerror5 lda #sumerr
C04A:38         (2) 176 gerror5 sec
C04B:60         (6) 177 rts
C04C:          (2) 178 *
C04C:          116 include pc.main

```

```

CD4C:      2 *
CD4C:      3 *
CD4C:      4 Entry equ *
CD4C:      5 bcc bentry
CD4C:      6 jmp bootcode
CD4E:      7 *
CD51:      8 * X is still set to slot number.
CD51:      9 *
CD51:      10 bentry equ *
CD51:      11 *
CD51:      12 lda #01000000
CD51:      13 lda #01000000
CD51:      14 trb ProfFlag+5
CD56:      15 atentry equ *
CD56:      16 *
CD56:      17 cld
CD56:      18 txa
CD56:      19 tay
CD59:      20 *
CD59:      21 * If this is a PC call, then get the address of the parm table
CD59:      22 *
CD59:      23 lda ProfFlag,y
CD59:      24 bml noplay
CD59:      25 *
CD59:      26 pla
CD59:      27 sta SHTempX,y
CD59:      28 cld
CD59:      29 adc #3
CD59:      30 tax
CD59:      31 pla
CD59:      32 sta SHTempY,y
CD59:      33 adc #0
CD59:      34 pha
CD59:      35 txa
CD59:      36 pha
CD59:      37 *
CD59:      38 noplay equ *
CD59:      39 *
CD59:      40 * On the //c, it is important to have the Disk // enable lines
CD59:      41 * off for as long as possible before using the IWM (phases,
CD59:      42 * /MREQ lines). Wait here 'til the Disk // motors are off.
CD59:      43 *
CD59:      44 jsr WaitIMOff
CD59:      45 *
CD59:      46 * We can't tolerate ints in most of the code, so disable
CD59:      47 *
CD59:      48 php
CD59:      49 sei
CD59:      50 *
CD59:      51 * Preserve the zero page work area
CD59:      52 *
CD59:      53 ldx #ZPSize-1
CD59:      54 pzp lda ZeroPage,x
CD59:      55 pha
CD59:      56 dex
CD59:      57 bpl pzp
CD59:      58 *
CD59:      59 * Okay, we're safe... now it's all right to store in zero page

```

```

CD7C:      60 *
CD7C:      61 sty Slot
CD7E:      62 *
CD7E:      63 *
CD7E:      64 * Now map any ProDOS unit references to our sequential ones.
CD7E:      65 * The method is bizzare and magicians never reveal their secrets.
CD7E:      66 *
CD7E:      67 allset equ *
CD7E:      68 lda CMDUnit
CD7E:      69 rol a
CD7E:      70 php
CD7E:      71 rol a
CD7E:      72 rol a
CD7E:      73 php
CD7E:      74 and #3210X767
CD7E:      75 and #00000011
CD7E:      76 eor #00000010
CD7E:      77 cpy #4
CD7E:      78 bge allset1
CD7E:      79 eor #00000010
CD7E:      80 allset1 tax
CD7E:      81 inx
CD7E:      82 stx CMDUnit
CD7E:      83 *
CD7E:      84 * Now if this is through the MLI xface, gotta copy stuff into the
CD7E:      85 * send buffer from the parameter list.
CD7E:      86 *
CD7E:      87 lda ProfFlag,y
CD7E:      88 bpl darnit
CD7E:      89 jmp skipcopy
CD7E:      90 *
CD7E:      91 * Get the address of the in-line parameter table
CD7E:      92 *
CD7E:      93 darnit equ *
CD7E:      94 lda SHTempX,y
CD7E:      95 sta buffer
CD7E:      96 lda SHTempY,y
CD7E:      97 sta buffer+1
CD7E:      98 *
CD7E:      99 * Now pull out the command code, and the address of the parameters.
CD7E:      100 *
CD7E:      101 ldy #1
CD7E:      102 lda (buffer),y
CD7E:      103 sta cmdcode
CD7E:      104 iny
CD7E:      105 lda (buffer),y
CD7E:      106 tax
CD7E:      107 iny
CD7E:      108 lda (buffer),y
CD7E:      109 sta buffer+1
CD7E:      110 stx buffer
CD7E:      111 *
CD7E:      112 * Now buffer points to parmlist
CD7E:      113 * Check command type, and pidgeonhole the parmlist length
CD7E:      114 *
CD7E:      115 lda #BadCmd
CD7E:      116 ldx cmdcode
CD7E:      117 cpx #SA

```

;Only valid codes are 0-9

```

CD00:90 03 CD02(3) 118 blt noeh
CD0F:4C 17 CF (3) 119 ErrorHitch jmp Error
CD02:      CD02 (3) 120 noeh equ *
CD02:A0 00 (2) 121 ldy #0
CD04:B1 54 (5) 122 lda (buffer),y
CD06:85 5A (3) 123 sta unit
CD08:      CD08 (3) 124 *
CD08:      CD08 (3) 125 * Now copy the bytes
CD08:      CD08 (3) 126 *
CD08:      CD08 (3) 127 okayent equ *
CD08:      CD08 (2) 128 ldy #<maxlength-1 ; Always copy the maximum
CD0A:      CD0A (5) 129 copyloop equ *
CD0A:B1 54 (5) 130 lda (buffer),y ; Pull it out of their hat
CD0C:99 42 00 (5) 131 sta cndcode,y ; Stuff it into mine
CD0F:88 (2) 132 dey ; Copy 'em all
CD0A:00 F8 CD0A(3) 133 bne copyloop
CD02:      CD02 (3) 134 *
CD02:      CD02 (3) 135 * Okay. The caller of the PC could be making one of three calls
CD02:      CD02 (3) 136 * with a unit number of $00, Control, Init or Status. Check for
CD02:      CD02 (3) 137 * these and do what is appropriate.
CD02:      CD02 (3) 138 *
CD02:A5 43 (3) 139 lda CNDUnit
CD04:D0 6A CE40(3) 140 bne skipcopy ; Never mind
CD06:      CD06 (3) 141 *
CD06:      CD06 (3) 142 * Check the parameter count for this call to unit#0
CD06:      CD06 (3) 143 *
CD06:A6 42 (3) 144 ldx CNDCode
CD08:BD 8E CF (4) 145 lda paramtab,x ; Get the length this command
CD0B:29 7F (2) 146 and #$FF ; Force 0 -> MSB
CD0D:A8 (2) 147 tay ; Bang on
CD0E:A9 04 (2) 148 lda #BadCnt ; Antic bad count
CD0F:C4 5A (3) 149 cpy unit ; User's pcount is currently here
CD02:D0 DB CD0F(3) 150 bne ErrorHitch ; What a baby!
CD04:      CD04 (3) 151 *
CD04:      CD04 (3) 152 * Now service one of the three commands
CD04:      CD04 (3) 153 *
CD04:      CD04 (2) 154 cpx #InitCMD
CD06:D0 0A CD02(3) 155 bne noInit ; Not an Init call
CD08:A9 00 (2) 156 lda #PowerReset ; Just like powerup or reset key(//c)
CD0A:98 CF (6) 157 jsr AssignID ; Do a reset cycle
CD0D:A9 00 (2) 158 ldx Aokay ; No error allowed
CD0F:4C 39 CF (3) 159 jmp sa2
CD02:      CD02 (2) 160 *
CD02:      CD02 (2) 161 notinit txa ; Equiv to 'cpx $StatusCMD'
CD03:D0 24 CE19(3) 162 bne maybectl
CD05:      CD05 (2) 163 *
CD05:A9 21 (2) 164 lda #BadCtl
CD07:A6 46 (3) 165 ldx CNDCode ; Antic a non zero stat code
CD09:D0 C4 CD0F(3) 166 bne ErrorHitch ; Stat unit#0 can only be code=0
CD0B:      CD0B (2) 167 *
CD0B:8A (2) 168 txa
CD0C:A6 58 (3) 169 ldx slot ; Equiv to 'lda #0'
CD0E:A0 07 (2) 170 ldy #7
CD00:91 44 (6) 171 minl ; Clear some space
CD02:88 CE00(3) 172 dey
CD03:D0 FB CE00(3) 173 bne minl
CD05:      CD05 (2) 174 *
CD05:BD F9 06 (4) 175 lda NumDevices,x

```

```

CD08:91 44 (6) 176 sta (CNDBuffer),y ; Stick it where they want it
CD0A:C8 (2) 177 iny
CD0B:      CD0B (2) 178 *
CD0B:A0 F9 04 (4) 179 lda $F9 ; //c Port 1 interrupt status
CD0C:      CD0C (2) 180 *
CD0C:91 44 (6) 181 sta (CNDBuffer),y ; Store PC interrupt status
CD0E:      CD0E (2) 182 *
CD0E:A9 08 (2) 183 lda #8 ; A, Y has 0000; # bytes status
CD0F:88 (2) 184 dey ; Skip down (up) with no error
CD13:20 F0 CF (6) 185 jsr squirrel
CD16:      CD16 (2) 186 *
CD16:4C ED CD CE19 (3) 187 jmp Aokay
CD19:      CD19 (2) 188 maybectl equ *
CD19:C9 04 CE28(3) 189 cpx #ControlCMD
CD1B:D0 0B CE28(3) 190 bne BUnit ; Unit #0 was a bad one
CD1D:      CD1D (2) 191 *
CD1D:A6 46 (3) 192 ldx CNDCode ; We allow two control calls for Unit#0
CD1F:F0 0B CE2C(3) 193 beq enabint ; 0 means enable interrupts
CD21:CA (2) 194 dex ; 1 means disable interrupts
CD22:F0 14 CE38(3) 195 beq disabint
CD24:A9 21 (2) 196 lda #badctl
CD26:      CD26 (2) 197 ErrorHitch2 equ *
CD26:D0 97 CD0F(3) 198 bne ErrorHitch ; No other codes allowed
CD28:      CD28 (2) 199 *
CD28:      CD28 (2) 200 BUnit equ *
CD28:8A 11 (2) 201 lda #BadUnit ; Only certain calls can have Unit#0
CD2A:D0 93 CD0F(3) 202 bne ErrorHitch ; Branch always
CD2C:      CD2C (2) 203 *
CD2C:      CD2C (2) 204 enabint equ *
CD2C:A9 C0 (2) 205 lda #$C0
CD2E:8D F9 05 (4) 206 sta $F9
CD31:A9 0F (2) 207 lda #F0F
CD33:0C 9A C0 (6) 208 tsb $C09A
CD36:D0 05 CE3D(3) 209 bne aokayhitch
CD38:      CD38 (2) 210 *
CD38:      CD38 (2) 211 disabint equ *
CD38:A9 01 (2) 212 lda #F01
CD3A:1C 9A C0 (6) 213 trb $C09A
CD3D:4C ED CD CE3D(3) 214 aokayhitch jmp AOkay
CD40:      CD40 (2) 215 *
CD40:      CD40 (2) 216 *
CD40:      CD40 (2) 217 * Okay, everything's all groovy. ProbOS re-enters here.
CD40:      CD40 (2) 218 * Check Unit number to be sure there is a corresponding device
CD40:      CD40 (2) 219 *
CD40:      CD40 (2) 220 skipcopy equ *
CD40:A9 28 (2) 221 lda #Bdrive ; Anticipate bad unit number
CD42:A4 58 (3) 222 ldy slot
CD44:BE F9 06 (4) 223 ldx NumDevices,y
CD47:84 43 (3) 224 cpx CNDUnit
CD49:90 DB CE26(3) 225 blt ErrorHitch2 ; Safe- If C clr then Z is clr
CD4B:      CD4B (2) 226 *
CD4B:      CD4B (2) 227 * Set buffer and bytecount in anticipation of the inevitable SendTrack.
CD4B:      CD4B (2) 228 *
CD4B:      CD4B (2) 229 lda #<maxlength
CD4D:85 40 (3) 230 sta bytecount1
CD4F:A9 00 (2) 231 lda #<maxlength
CD51:85 4E (3) 232 sta bytecount
CD53:85 55 (3) 233 sta buffer+1

```

```

07 PC.MAIN      Protocol Converter / CBus Driver      20-OCT-86 06:29 PAGE 39

CE55:A9 42      (2) 234      lda    $>cmdcode
CE57:85 54      (3) 235      sta    buffer
CE59:          (3) 236 *      ;If it's a PC call, omit the next two steps
CE59:          (3) 237 *
CE59:          (3) 238 *
CE59:A6 58      (3) 239      ldx    slot
CE5B:BD 73 04   (4) 240      lda    ProFlag,x      ;Is it a call from ProDOS?
CE5E:10 13      CE73(3) 241      bpl    notstat      ;=> Statcode already set...
CE60:          (2) 242 *
CE60:          243 *      ;Need to generate a parameter count for a ProDOS call
CE60:          244 *
CE60:          245 *
CE60:A6 42      (3) 245      ldx    CMDCode
CE62:BD 8E CF   (4) 246      lda    ParamTab,x
CE65:29 7E     (2) 247      and    #STF
CE67:85 5A      (3) 248      sta    Unit
CE69:          (2) 249 *
CE69:          250 *      ;ProDOS always needs the highest blockno byte zeroed
CE69:          251 *
CE69:A9 00      (2) 252      lda    #0
CE6B:85 48      (3) 253      sta    CMDBlock
CE6D:          (2) 254 *
CE6D:          255 *      ;If this is a ProDOS status call, set stat code to zero
CE6D:          256 *
CE6D:          257 *
CE6D:A5 42      (3) 257      ldx    CMDCode
CE6F:00 02      CE73(3) 258      bne    notstat      ;=> Not status so forget it
CE71:          (3) 259 *      ;lda    #DeviceStat ;A is already zero
CE71:85 46      (3) 260      sta    CMDCode      ;Store in command table
CE73:          (2) 261 *
CE73:          262 *      ;Okay, finally send over the damn command
CE73:          263 *
CE73:          CE73      264      notstat equ *
CE73:A5 5A      (3) 265      lda    Unit
CE75:A6 43      (3) 266      ldx    CmdPcount      ;Swap the Paramcount & unit#
CE77:86 5A      (3) 267      stx    Unit
CE79:85 43      (3) 268      sta    CMDPcount      ;Now they're correct
CE7B:          (2) 269 *
CE7B:A9 80      (2) 270      lda    #cmdmark
CE7D:85 5B      (3) 271      sta    WPacketType
CE7F:          (3) 272 *
CE7F:20 87 CA   (6) 273      jsr    ClrPhases      ;Bring all phases off for Quark
CE82:          (2) 274 *
CE82:          275      jsr    SendPack
CE85:B0 46      CECD(3) 276      bcs    behitch      ;If not okay, skip to bus error
CE87:          (3) 277 *
CE87:          278 *      ;Now copy over the buffer address for any data xfer.
CE87:          279 *
CE87:          (3) 280      lda    CMDBuffer
CE89:85 54      (3) 281      sta    buffer
CE8B:A5 45      (3) 282      lda    CMDBuffer+1
CE8D:85 55      (3) 283      sta    buffer+1
CE8F:          (2) 284 *
CE8F:          285 *      ;Now for some commands, we have to send over a packet of data, too.
CE8F:          286 *      ;See if this command is one of THOSE.
CE8F:          287 *
CE8F:A6 42      (3) 288      ldx    cmdcode
CE91:BD 8E CF   (4) 289      lda    paramtab,x
CE94:10 3B      CEED(3) 290      bpl    noxtrasend      ;Encoded in top bit
CE96:          (3) 291 *

```

```

07 PC.MAIN      Protocol Converter / CBus Driver      20-OCT-86 06:29 PAGE 40

CE96:          292 *      ;The buffer address and bytecount depend on the call type.
CE96:          293 *
CE96:E0 04      (2) 294      cpx    #ControlCmd
CE98:D0 18      CE93(3) 295      bne    NOControl
CE9A:          (2) 296 *
CE9A:          297 *      ;In the case of control, bytecount:=(buffer)
CE9A:          298 *      ;and buffer := buffer+2
CE9A:          299 *
CE9A:A0 01      (2) 300      ldy    #1
CE9C:B1 54      (5) 301      lda    (buffer),y      ;Get hi order bytecount
CE9E:8A        (2) 302      tax
CE9F:86        (2) 303      dey
CEA0:B1 54      (5) 304      lda    (buffer),y
CEA2:48        (3) 305      pha
CEA3:18        (2) 306      clc
CEA4:A9 02      (2) 307      lda    #2
CEA6:65 54      (3) 308      adc    buffer
CEA8:85 54      (3) 309      sta    buffer
CEAA:68        (4) 310      pla
CEAB:90 13      CECD(3) 311      bcc    secondsend      ;Get back Lo order bytecount
CEAD:E6 55      (5) 312      inc    buffer+1      ;Skip hi ord increment
CEAF:4C 00 CE   (3) 313      jmp    secondsend      ;Skip to store bytecount
CEB2:          (3) 314 *
CEB2:          CEB2      315      NOControl equ *
CEB2:E0 02      (2) 316      cpx    #WriteCMD      ;Check for a writeblock
CEB4:D0 06      CEBC(3) 317      bne    NOWBlock      ;Must be control or write
CEB6:          (3) 318 *
CEB6:          319 *      ;In the case of WriteBlock, the length is 512 and the buffer
CEB6:          320 *      ;address is at buffer in the command table
CEB6:          321 *
CEB6:A9 00      (2) 322      lda    #0
CEB8:A2 02      (2) 323      ldx    #2
CEBA:D0 04      CECD(3) 324      bne    secondsend
CEBC:          (3) 325 *
CEBC:          326 *      ;For FileWrite, the buffer address is at CMDbuffer
CEBC:          327 *      ;and the length is at CMDblock.
CEBC:          328 *
CEBC:          CEBC      329      NOWBlock equ *
CEBC:A6 47      (3) 330      ldx    CMDBlock
CEBC:A5 46      (3) 331      lda    CMDBlock+1
CECE:          (3) 332 *
CECE:          CECE      333      secondsend equ *
CECE:          (3) 334      stx    bytecount
CECE:          (3) 335      sta    bytecount+1
CECE:          (3) 336 *
CECE:A9 82      (2) 337      lda    #datamark
CECE:85 58      (3) 338      sta    WPacketType      ;Identify this as a data packet
CECE:          (3) 339 *
CECE:          CECE      340      jsr    SendData
CECE:          (3) 341      bcc    noxtrasend
CECE:          CECD      342      behitch equ *
CECE:          (2) 343      lda    #BusErr      ;This is the bus error latch
CECE:D0 46      CE17(3) 344      bne    Error
CED1:          (3) 345 *
CED1:          346 *      ;On ProDOS status call, we've got to point the buffer pointer
CED1:          347 *      ;correctly to zero page... it's the only case special case
CED1:          348 *      ;(on Write, format and Control no data comes back).
CED1:          349 *

```

```

CED1: 350 movtrasequ *
CED1:A4 58 (3) 351 ldy slot
CED3:B9 73 04 (4) 352 lda ProFlag.y
CED6:10 0C CEE4(3) 353 bpl getresults
CED8:A5 42 (3) 354 lda cmcode
CEDA:D0 08 CEE4(3) 355 bne getresults
CEDC: 356 *
CEDC:A9 45 (2) 357 lda #CMBDufferh ;Want status in these four
CEDC:A2 00 (2) 358 ldx #CMBDufferh
CEDC:85 54 (3) 359 sta buffer
CED2:86 55 (3) 360 stx buffer+1
CEEA: 361 *
CEEA: 362 * Please to be calling ReceivePack
CEEA: 363 *
CEEA: 364 getresults equ *
CEEA:20 30 CB (6) 365 jsr RecPack ;Get status byte (maybe read data too)
CEEA:B0 24 CEEC(3) 366 bcs behitck
CEE9: 367 *
CEE9: 368 * Figure how many bytes were sent and put that in X,Y temps
CEE9: 369 *
CEE9:20 51 CC (6) 370 jsr Rvcount ;Do the times 7...
CEE9:F0 F0 CF (6) 371 jsr squirrel ;Store away count in SRRxggs
CEE: 372 *
CEE: 373 * For the ProDOS status call, we've got to look at the status byte
CEE: 374 * returned and return a DIP error if appropriate. Also overwrite
CEE: 375 * the X,Y temps with # blocks if this is a ProDOS Stat call.
CEE: 376 *
CEE: 377 lda CMCode ;Is it a ProDOS status call
CEE:A5 42 (3) 378 bne noerror
CEE:F0 22 CF15(3) 379 ldx slot
CEE3:A6 58 (3) 380 lda ProFlag.x
CEE5:B0 73 04 (4) 380 bpl noerror
CEE8:10 1B CF15(3) 381 *
CEFA: 382 *
CEFA:A5 46 (3) 383 lda CMBDBlockl
CEFC:90 F3 05 (5) 384 sta SRRtemp.x
CEE3:A5 47 (3) 385 lda CMBDBlockh
CF01:9D 73 06 (3) 386 sta SRRtemp.y
CF04: 387 *
CF04:A5 45 (3) 388 lda CMBDufferh ;Check status byte
CF06:4A (2) 389 lsr a
CF07:4A (2) 390 lsr a
CF08:4A (2) 391 lsr a
CF09:90 04 CF0F(3) 392 bcc ChOfflin ;no error, go check off line
CF0B:A9 2B (2) 393 lda #WriteProt ;else set WPROT error
CF0D:80 08 CF17(3) 394 bra error
CF0F: 395 ChOfflin equ *
CF0F:4A (2) 396 lsr a
CF0F:4A (2) 397 lsr a
CF11:A9 2F (2) 398 lda #Offline
CF13:90 02 CF17(3) 399 bcc error
CF15: 400 *
CF15: 401 * Now it's time to think about returning to the caller
CF15: 402 * Remember that ProDOS doesn't want to know about soft errors,
CF15: 403 * only fatal ones. If this is a ProDOS call, and the soft error
CF15: 404 * bit in the statbyte is set, there IS NO error (statbyte is
CF15: 405 * cleared). Also, ProDOS wants only I/O, Write Protect, NO Device,
CF15: 406 * Offline. If any other hard error comes from the device
CF15: 407 * on a ProDOS call, map it to an I/O Error. (Gross me out.)

```

```

CF15: 408 *
CF15: 409 noerror equ *
CF15:A5 4D (3) 410 lda statbyte
CF17: 411 Error equ *
CF17:A4 58 (3) 412 ldy Slot
CF19:99 F3 04 (5) 413 sta Retry.Y ;Need access to screenholes
CF1C:AA (2) 414 tax ;Keep unadulterated error in shole
CF1D:F0 1A CF39(3) 415 beq sa2 ;Set the Z flag
CF1F: 416 * ;Special case the zero
CF1F:BE 73 04 (4) 417 ldx ProFlag.y ;Set N to ProDOS call or not
CF22:10 15 CF39(3) 418 bpl sa2 ;If PC call, no mapping occurs
CF24: 419 *
CF24:A2 00 (2) 420 ldx #0 ;Assume a soft error
CF26:C9 40 (2) 421 cmp #401000000 ;Soft error check
CF28:B0 0E CF38(3) 422 bge storeaway ;If $40 or bigger, map to zero
CF2A: 423 *
CF2A:A2 27 (2) 424 ldx #IOError ;Now anticipate ProDOS I/O error
CF2C:C9 2B (2) 425 cmp #WriteProt ;OK to return Write Protect
CF2E:F0 09 CF39(3) 426 beq sa2
CF30:C9 2B (2) 427 cmp #NoDrive
CF32:F0 05 CF39(3) 428 beq sa2
CF34:C9 2F (2) 429 cmp #Offline
CF36:F0 01 CF39(3) 430 beq sa2
CF38: 431 *
CF38: 432 storeaway equ * ;Use the default value
CF38: 433 *
CF38:8A (2) 433 tax
CF39: 434 sa2 equ *
CF39:A5 58 (3) 435 ldy Slot
CF3B:99 73 05 (5) 436 sta SRRtemp.y
CF3E: 437 *
CF3E: 438 * If this is the //c version, we need to reset the IM to its
CF3E: 439 * former disk // state. This is done by setting the mode register
CF3E: 440 * to a little known (and less documented) mode which speeds up the
CF3E: 441 * internal motor timeout. When the motor enable has timed out, the
CF3E: 442 * mode can be set back to zero. This method is necessary because
CF3E: 443 * if the timer is enabled within the timeout period, the motor on a
CF3E: 444 * Rev A IM pops on for the full timeout period (since mode changes
CF3E: 445 * are disabled when the motor is on. It's bizzarra. Blame Mac.
CF3E:AD 2B C0 (4) 446 lda monclr+$60 ;Motor off
CF41:2C D0 C0 (4) 447 bit l6set+$60 ;Into mode reg access mode
CF44:A9 2B C0 (4) 448 lda #2B ;This is the magic "speed up" value
CF46:8D EF C0 (4) 449 sta l7set+$60 ;Throw into mode register
CF49:EA (2) 450 nop ;You're supposed to wait a while
CF4B:EA (2) 451 nop
CF4C:EA (2) 452 nop
CF4D: 453 *
CF4D: 454 waitoff equ *
CF4D:AD EF C0 (4) 455 lda l7clr+$60 ;Wait 'til motor off
CF50:29 20 (2) 456 and #520
CF52:D0 F9 CF4D(3) 457 bne waitoff
CF54:A0 00 (2) 458 ldy #0 ;Now set the reg back to $00
CF56:A2 60 (2) 459 ldx #560 ;IM's in slot 6
CF58:20 20 CC (6) 460 jsr SetIMode
CF5B:AD EC C0 (4) 461 lda l6clr+$60
CF5E:AD E2 C0 (4) 462 lda l6clr+$60
CF61:AD E6 C0 (4) 463 lda l6tblclr+$60
CF64:A4 58 (3) 464 ldy Slot ;Need Slot in Y
CF66: 465 *

```

```

CF66:      466 * Now, restore our zero page area.
CF66:      467 *
CF66:A2 00 (2) 468 ldx #0
CF68:68 (4) 469 rzp pla
CF69:95 40 (4) 470 sta zeropage,x
CF6A:E8 (4) 471 lnx
CF6C:E0 1C (2) 472 cpx #0Size
CF6E:90 F8 CF68 (3) 473 blt rzp
CF70:      474 *
CF70:      475 * We're into the stretch! Restore interrupt mask, load X, Y,
CF70:      476 * and A and set the carry if the error byte is non-zero.
CF70:      477 *
CF70:      478 pip
CF71:B9 F3 05 (4) 479 ldx STTemp,x
CF74:AA (2) 480 tax
CF75:B9 73 05 (4) 481 ldx STTemp1,y
CF78:48 (3) 482 pha
CF79:B9 73 06 (4) 483 ldx STTemp,y
CF7C:A8 (2) 484 tay
CF7E:68 (4) 485 clc
CF7F:F0 01 CF82 (3) 487 beq finalskip
CF81:38 (2) 488 sec
CF82:      489 finalskip equ *
CF82:      490 *
CF82:08 (3) 491 pip
CF83:7C 78 04 (4) 492 bit ProFlag+5
CF86:70 04 CF8C (3) 493 bvs ick1
CF88:28 (4) 494 pip
CF89:4C 84 C7 (3) 495 jmp SWRTS2
CF8C:      496 ick1 equ *
CF8C:28 (4) 497 pip
CF8D:60 (6) 498 rts
CF8E:      499 *
CF8E:      500 *
CF8E:      501 parmtab equ *
CF8E:03 (5) 502 dfb $00000011
CF8F:03 (5) 503 dfb $00000011
CF90:83 (5) 504 dfb $10000011
CF91:01 (5) 505 dfb $00000001
CF92:83 (5) 506 dfb $00000011
CF93:01 (5) 507 dfb $00000001
CF94:01 (5) 508 dfb $00000001
CF95:01 (5) 509 dfb $00000001
CF96:03 (5) 510 dfb $00000011
CF97:83 (5) 511 dfb $10000011
CF98:      512 *
CF98:      513 *

```

```

CF98:      515 *
CF98:      516 AssignID equ *
CF98:48 (3) 517 pha
CF99:20 5D CA (6) 518 jsr resetchain
CF9C:68 (4) 519 pla
CF9D:AA (2) 520 tax
CF9E:      521 *
CF9E:      522 * Save the command code, unit, and init code
CF9E:      523 * 'cause we'll trample 'em.
CF9E:      524 *
CF9E:A5 42 (3) 525 ldx CMDCode
CF9F:48 (3) 526 pha
CF9F:A5 43 (3) 527 ldx CMDPCount
CF9F:48 (3) 528 pha
CF9F:A5 46 (3) 529 ldx CMDSCode
CF9F:48 (3) 530 pha
CF9F:86 46 (3) 531 stx CMDSCode
CF9F:      532 *
CF9F:      533 * Set up to send DefID command packets
CF9F:      534 *
CF9F:A9 05 (2) 535 ldx #InitCmd
CF9F:A9 00 (3) 536 sta CMDCode
CF9F:85 42 (2) 537 ldx #0
CF9F:85 5A (3) 538 sta Unit
CF9F:A9 02 (2) 539 ldx #2
CF9F:      540 *
CF9F:      541 * Point the buffer pointer
CF9F:      542 *
CF9F:85 42 (2) 543 ldx #CMDCode
CF9F:85 54 (3) 544 sta buffer
CF9F:A9 00 (2) 546 ldx #CMDCode
CF9F:85 55 (3) 547 sta buffer+1
CF9F:A9 80 (2) 548 ldx #cmdmark
CF9F:85 5B (3) 549 sta WPacketType
CF9F:      550 *
CF9F:      551 *
CF9F:      552 *
CF9F:      553 * Send an ID for the next device in the chain
CF9F:      554 *
CF9F:      555 mordevices equ *
CF9F:86 5A (5) 556 inc Unit
CF9F:A9 09 (2) 557 ldx #cmdlength
CF9F:85 4D (3) 558 sta bytecount1
CF9F:A9 00 (2) 559 ldx #cmdlength
CF9F:85 4E (3) 560 sta bytecount
CF9F:      561 *
CF9F:      562 *
CF9F:      563 *
CF9F:      564 *
CF9F:86 5A (5) 565 dec Unit
CF9F:4C DE CF (3) 566 jmp mdev1
CF9F:      567 *
CF9F:86 mdev2 (6) 568 mdev2 jsr ReceivePack
CF9F:A5 4D (3) 569 ldx statbyte
CF9F:F0 E5 CF9C (3) 570 beq mordevices
CF9F:      571 *
CF9F:      572 * Okay, we done last device. Squirrel away the number of devices.

```

C908 ACH81  
 C909 ALLSET1  
 C910 ALLSET1  
 C911 AUTOSCAN  
 C912 AUTOSCAN  
 C913 AUTOSCAN  
 C914 AUTOSCAN  
 C915 AUTOSCAN  
 C916 AUTOSCAN  
 C917 AUTOSCAN  
 C918 AUTOSCAN  
 C919 AUTOSCAN  
 C920 AUTOSCAN  
 C921 AUTOSCAN  
 C922 AUTOSCAN  
 C923 AUTOSCAN  
 C924 AUTOSCAN  
 C925 AUTOSCAN  
 C926 AUTOSCAN  
 C927 AUTOSCAN  
 C928 AUTOSCAN  
 C929 AUTOSCAN  
 C930 AUTOSCAN  
 C931 AUTOSCAN  
 C932 AUTOSCAN  
 C933 AUTOSCAN  
 C934 AUTOSCAN  
 C935 AUTOSCAN  
 C936 AUTOSCAN  
 C937 AUTOSCAN  
 C938 AUTOSCAN  
 C939 AUTOSCAN  
 C940 AUTOSCAN  
 C941 AUTOSCAN  
 C942 AUTOSCAN  
 C943 AUTOSCAN  
 C944 AUTOSCAN  
 C945 AUTOSCAN  
 C946 AUTOSCAN  
 C947 AUTOSCAN  
 C948 AUTOSCAN  
 C949 AUTOSCAN  
 C950 AUTOSCAN  
 C951 AUTOSCAN  
 C952 AUTOSCAN  
 C953 AUTOSCAN  
 C954 AUTOSCAN  
 C955 AUTOSCAN  
 C956 AUTOSCAN  
 C957 AUTOSCAN  
 C958 AUTOSCAN  
 C959 AUTOSCAN  
 C960 AUTOSCAN  
 C961 AUTOSCAN  
 C962 AUTOSCAN  
 C963 AUTOSCAN  
 C964 AUTOSCAN  
 C965 AUTOSCAN  
 C966 AUTOSCAN  
 C967 AUTOSCAN  
 C968 AUTOSCAN  
 C969 AUTOSCAN  
 C970 AUTOSCAN  
 C971 AUTOSCAN  
 C972 AUTOSCAN  
 C973 AUTOSCAN  
 C974 AUTOSCAN  
 C975 AUTOSCAN  
 C976 AUTOSCAN  
 C977 AUTOSCAN  
 C978 AUTOSCAN  
 C979 AUTOSCAN  
 C980 AUTOSCAN  
 C981 AUTOSCAN  
 C982 AUTOSCAN  
 C983 AUTOSCAN  
 C984 AUTOSCAN  
 C985 AUTOSCAN  
 C986 AUTOSCAN  
 C987 AUTOSCAN  
 C988 AUTOSCAN  
 C989 AUTOSCAN  
 C990 AUTOSCAN  
 C991 AUTOSCAN  
 C992 AUTOSCAN  
 C993 AUTOSCAN  
 C994 AUTOSCAN  
 C995 AUTOSCAN  
 C996 AUTOSCAN  
 C997 AUTOSCAN  
 C998 AUTOSCAN  
 C999 AUTOSCAN  
 C1000 AUTOSCAN

CFE0: 573 \*  
 CFE1: 574 mdev1 lda Unit  
 CFE2: 575 ldy slot  
 CFE3: 576 sta NumDevices.y :Devices out there  
 CFE4: 577 \*  
 CFE5: 578 \* Recover the scrambled ProDOS parms  
 CFE6: 579 \*  
 CFE7: 580 pla CMOStCode  
 CFE8: 581 sta sta CMOStCode  
 CFE9: 582 pla CMOStCode  
 CFEA: 583 sta CMOStCode  
 CFEB: 584 pla CMOStCode  
 CFEC: 585 sta CMOStCode  
 CFED: 586 \*  
 CFEE: 587 rts  
 CFEF: 588 \*  
 CFF0: 589 \*  
 CFF1: 590 squirrel equ \*  
 CFF2: 591 ldx slot  
 CFF3: 592 sta STempX,x  
 CFF4: 593 tya STempX,x  
 CFF5: 594 sta STempX,x  
 CFF6: 595 rts  
 CFF7: 596 \*  
 CFF8: 597 \*  
 CFF9: 598 \*  
 CFFA: 599 \*  
 CFFB: 599 \*  
 CFFC: 599 \*  
 CFFD: 599 \*  
 CFFE: 599 \*  
 CFFF: 599 \*  
 C1000: 599 \*

07 SYMBOL TABLE		20-OCT-86 06:29 PAGE 47		07 SYMBOL TABLE		20-OCT-86 06:29 PAGE 48	
3E93 SETVID	C9A7 SETXNO	C9A9 SHIF71	C9AA SHIF72	? 00 STATUSMD	? 00 SERVICESTAT	00 PCID2	00 LOCO
C9A8 SHIF73	C9A8 SHIF74	C9A9 SHIF75	C9A9 SHIF76	00 POWERRES1	? 01 LCL	01 RADCMD	? 01 SCGETDCH
0673 SHTMPX	C9A9 SKIP1	C9A9 SKIP2	C9A9 SKIP3	01 RADCMD	? 01 MONSWR	02 NOARK	? 02 WATECMD
C9A9 SKIP4	C9A9 SKIP5	C9A9 SKIP6	C9A9 SKIP7	? 02 SCGETINFO	? 03 FORMATCMD	? 03 SCGETINFO	? 04 WARESET
C9A9 SOB1	C9A9 SOB2	C9A9 SOB3	C9A9 SOB4	04 CONTROLCHD	07 IMMOC	? 08 BYTCMP	09 CMLLENGTH
? 67 SOFTERROR	C9A9 SP1L1	C9A9 SP1L2	C9A9 SP1L3	06 BUSERR	10 CSWERR	? 10 SWASK1	0011 BASGLN
C9A9 SSR	C9A9 SSD	C9A9 SSS	C9A9 SSS1	0A BSTO2	1C ZPSIZE	? 1E STATO	? 1F NOINT
C9A9 START1	C9A9 START2	C9A9 START3	C9A9 START4	11 RADUNIT	21 RADCTL	? 22 RADCTLPAH	? 24 CB
C9A9 START5	C9A9 START6	C9A9 START7	C9A9 START8	? 25 CV	27 IOERR	28 MODRIVE	2B WRITERPOT
C9A9 START9	C9A9 START10	C9A9 START11	C9A9 START12	? 2D RADLOCK	2F OFFLINE	32 BSTO1	40 ZEROPAGE
C9A9 START13	C9A9 START14	C9A9 START15	C9A9 START16	? 40 BUSG0	40 CHECKSUM	40 SFT	41 TOPBITS
C9A9 START17	C9A9 START18	C9A9 START19	C9A9 START20	42 CMDCODE	43 CMDCOUNT	43 CMDCOUNT	44 CMDBUFFER
C9A9 START21	C9A9 START22	C9A9 START23	C9A9 START24	44 CMDBUFFERL	45 CMDBUFFERH	46 CMDCODE	? 46 CMDBLOCK
C9A9 START25	C9A9 START26	C9A9 START27	C9A9 START28	? 4A CMDBLOCK	47 CMDBLOCKH	48 CMDBLOCKS	? 49 CMDBAREL
C9A9 START29	C9A9 START30	C9A9 START31	C9A9 START32	48 CMDBLOCKL	4B CMDBUF	4B CMDBUF	4C ODBYTE1
C9A9 START33	C9A9 START34	C9A9 START35	C9A9 START36	40 NEXT1	40 BYTCOUNTL	40 NEXT2	40 STATBYTE
C9A9 START37	C9A9 START38	C9A9 START39	C9A9 START40	40 NEXT3	4F NEXT3	50 NEXT4	? 4E AUTYPE
C9A9 START41	C9A9 START42	C9A9 START43	C9A9 START44	? 4F PACKETTYPE	51 NEXT5	52 NEXT6	? 50 DEVICEID
C9A9 START45	C9A9 START46	C9A9 START47	C9A9 START48	? 51 NEXT7	54 BUFFER	56 AUXPR	? 52 POINTER
C9A9 START49	C9A9 START50	C9A9 START51	C9A9 START52	58 SLOT	59 TROD	59 TEMP	56 BUFFER2
C9A9 START53	C9A9 START54	C9A9 START55	C9A9 START56	58 WPACKETTYPE	60 TROFF	? 67 SOFTERROR	? 68 LASTONE
C9A9 START57	C9A9 START58	C9A9 START59	C9A9 START60	80 COMRESET	80 COMARK	? 81 STATMARK	82 DATAMARK
C9A9 START61	C9A9 START62	C9A9 START63	C9A9 START64	? 85 PBEVALUE	8F POIDRITTE	C3 PACKETREG	C8 PACKETEND
C9A9 START65	C9A9 START66	C9A9 START67	C9A9 START68	? FF PBEVALUE	? 0100 STACK	? 0101 VERSION	0473 PROFAG
C9A9 START69	C9A9 START70	C9A9 START71	C9A9 START72	0473 SCHOLES	0473 RETRY	0573 RETRY2	0573 SHTMP1
C9A9 START73	C9A9 START74	C9A9 START75	C9A9 START76	0573 SHTMPX	0673 SHTMPY	0678 SVACL	0679 NMDVEICES
C9A9 START77	C9A9 START78	C9A9 START79	C9A9 START80	0778 SVACH	0708 BOOTSCN	0708 MSLOT	0888 RC1
C9A9 START81	C9A9 START82	C9A9 START83	C9A9 START84	C9A9 RECLR	C9A9 IM	C9A9 RESET	C9A9 CALCLR
C9A9 START85	C9A9 START86	C9A9 START87	C9A9 START88	C9A9 CAIS1	C9A9 CAZCLR	C9A9 CAZSET	C9A9 LSTRCLR
C9A9 START89	C9A9 START90	C9A9 START91	C9A9 START92	C9A9 LSTRSET	C9A9 MONCLR	C9A9 MONSET	? C9A9 ENABE1
C9A9 START93	C9A9 START94	C9A9 START95	C9A9 START96	C9A9 ENABE2	C9A9 L6CLR	C9A9 L6SET	C9A9 L7CLR
C9A9 START97	C9A9 START98	C9A9 START99	C9A9 START100	C9A9 L7SET	? C500 C500RG	C50A PRODCENTRY	C500 MLENTY
C9A9 START101	C9A9 START102	C9A9 START103	C9A9 START104	? C514 BOOTCLAS5	C521 BOOTC	C523 BOOTCODE	C52F RC1
C9A9 START105	C9A9 START106	C9A9 START107	C9A9 START108	C552 BOOTTAIL	C554 MORCHS	C55D COMA	C55F BOOTHG
C9A9 START109	C9A9 START110	C9A9 START111	C9A9 START112	C570 BOOTTAIL	C576 RESET	C578 RST1	C583 RCDE
C9A9 START113	C9A9 START114	C9A9 START115	C9A9 START116	? C58A CMLIST	C784 SWRTS2	C797 SWPROTO	C883 SENDONEPACK
C9A9 START117	C9A9 START118	C9A9 START119	C9A9 START120	C9A9 UBSY1	C9A9 CHAINUNBSY	C9A9 SSR	C9A9 SSD
C9A9 START121	C9A9 START122	C9A9 START123	C9A9 START124	C9A9 SOB1	C9A9 SOB2	C9A9 SOB3	C9A9 START
C9A9 START125	C9A9 START126	C9A9 START127	C9A9 START128	C9A9 ACHE1	C9A9 ACHE2	C9A9 SKIP2	C9A9 SKIP3
C9A9 START129	C9A9 START130	C9A9 START131	C9A9 START132	C9A9 SKIP4	C9A9 SKIP5	C9A9 SCH1	C9A9 SD7
C9A9 START133	C9A9 START134	C9A9 START135	C9A9 START136	C9A9 PATCH1	C9A9 PATCH2	C9A9 SD9	C9A9 SD10
C9A9 START137	C9A9 START138	C9A9 START139	C9A9 START140	C9A9 PREMBLE	C9A9 SYNCTAB	? C909 WASTE12	? C90C WASTE18
C9A9 START141	C9A9 START142	C9A9 START143	C9A9 START144	? C900 WASTE16	C90E WASTE14	? C90F WASTE12	C900 MARKERR
C9A9 START145	C9A9 START146	C9A9 START147	C9A9 START148	C9A9 RECIWEPACK	? C9A9 GRABSTATUS	C9A9 ROH1	C9A9 ROH2
C9A9 START149	C9A9 START150	C9A9 START151	C9A9 START152	? C9A9 ROH5	C9A9 ROH3	C9A9 START0	C9A9 START1
C9A9 START153	C9A9 START154	C9A9 START155	C9A9 START156	C9A9 G0B1	C9A9 START2	C9A9 SEND80	C9A9 SEND81
C9A9 START157	C9A9 START158	C9A9 START159	C9A9 START160	C9A9 RESETCHAIN	C9A9 YMSHAT	C9A9 OREMS	C9A9 SHIF1
C9A9 START161	C9A9 START162	C9A9 START163	C9A9 START164	C9A9 ENABECHAIN	C9A9 CLRPASHS	C9A9 SETXNO	C9A9 SHIF4
C9A9 START165	C9A9 START166	C9A9 START167	C9A9 START168	C9A9 SHIF2	C9A9 SHIF3	C9A9 SHIF4	C9A9 SENDDATA
C9A9 START169	C9A9 START170	C9A9 START171	C9A9 START172	C9A9 SDOUFF	C9A9 SENDPACK	C9A9 SENDPIL	? C9A9 ALTSENDPIL
C9A9 START173	C9A9 START174	C9A9 START175	C9A9 START176	C9A9 SP1L1	C9A9 SP1L2	C9A9 RECPACK	C9A9 RPK1
C9A9 START177	C9A9 START178	C9A9 START179	C9A9 START180	C9A9 SP1L3	C9A9 SP1L4	C9A9 PMO7TAB	C9A9 DIV7TAB
C9A9 START181	C9A9 START182	C9A9 START183	C9A9 START184	C9A9 MO7TAB	C9A9 PMO7TAB	C9A9 WRITERP	? C9A9 DIVIDE7
C9A9 START185	C9A9 START186	C9A9 START187	C9A9 START188	C9A9 SAPI	C9A9 DIVIDE3	C9A9 DIVIDE4	C9A9 DIVIDE5
C9A9 START189	C9A9 START190	C9A9 START191	C9A9 START192	C9A9 DIVIDE1	C9A9 DIVIDE2	? C9A9 PRECHECK	C9A9 DIVIDE6
C9A9 START193	C9A9 START194	C9A9 START195	C9A9 START196	? C9A9 DIVIDE2	C9A9 X0R1	C9A9 X0R2	C9A9 LASTPASS

```

CB06 XOR3
?CC01 SUN
CC20 SETMODE
CC3F WIMM1
CC57 T71
CC7D START35
CD34 ICFT15
CD48 CSERROR5
?CD56 AENTRY
CD90 ALISET1
?CD98 ORACWT
CE00 NIN1
CE2C ENABINT
CE73 NOTSTAT
CEED BEHITCH
CF15 NOERROR
CF4D WAITOT?
CF8E PARMC7AB
CF9F MDEV1
?FABA AUTOSCAN
?FE91 SETVID
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED 2157
** FREE SPACE PAGE COUNT 70

```

```

CB0D XOR4
CC03 SUN2
CC29 E12
CC4C WIMM2
CC70 T72
CD0A DONES
?CD37 LSTSYNALT5
CD4A CSERROR5
CD6F NOPLAY
CD9C DAWTIT
CDCA COPYLOOP
CE19 MAIBCTRL
CE38 DISABINT
CEB2 NOCONTROL
CED1 NOXTRASEND
CF17 ERROR
CF68 RFP
CF96 ASSIGNID
CF9F SQUIREL
?FC22 VTAB

```

```

?CBE2 DETOPBITS
CC0C SUN1
CC2D CAREFUL
CC31 RCVCOUNT
CC73 SLOTERPD
CD1F RDBA5
CD3A RDBA5
CD4C ENTRY
CD76 P?P
CD8F ERRORHITCH
CDED AORAY
CE26 ERRORHITCH2
CE3D AORAYHITCH
CEB3 MOWBLACK
CEB4 GETRESULTS
CF38 STOREMAY
CF82 FINALSKIP
CF84 MORDEVICES
CF84 CLEAROROMS
?FDED COUT

```

```

CEB9 GTROB
CC1F SUN3
CC36 WAITIMOFF
CC3A TIMES7
?CC73 START25
CDE ICFT5
CD44 NFENDERS5
CD51 BENTRY
?C07E ALLSET
C0C2 NOEB
C0F2 NOTINIT
CE28 BUNIT
CE40 SKIPCOPY
CEC0 SECONDSEND
CF0F CHKOFFLIN
CF39 SA2
CF8C ICK1
CFD8 MDEV2
?E000 RASIC
?FE89 SETRED

```

SOURCE FILE #01 =>ASM.S  
INCLUDE FILE #02 =>S\_DIAG1.SRC

```
0000:          1      1st on,vsym,asym
0000:          2      include s.diag1.src
```

```
02 S_DIAG1.SRC      slinky diagnostics      20-OCT-86 06:36 PAGE 2
0000: 2 *****
0000: 3 *
0000: 4 *      Internal Slinky Diagnostics
0000: 5 *
0000: 6 *****
0000:
0000: 8 *****
0000: 9 * written by Eric Larson      19 April 1985
0000: 10 * modified by Rich Williams      09 May 1985
0000: 11 * put into //c rom by Ray Chiang      20 Feb 1986
0000: 12 *****
0000:
0000: 14 *****
0000: 15 * on entry: y-reg has the value of sl.slot (screen hole offset)
0000: 16 *      x-reg has the value of sl.devno (hardware offset)
0000: 17 *      card size is in numbanks.y
0000: 18 *****
0000:
0000: 20 *****
0000: 21 *      equates
0000: 22 *****
0000: 23 mptr      equ $42
0000: 24 testnum      equ $00
0000: 25 compdata      equ $45
0000: 26 limit      equ $46
0000: 27 value      equ $47
0000: 28 loopcount      equ $49
0000: 29 cv      equ $25
0000: 30 dot      equ $AE
0000: 31 bell      equ $07
0000: 32 cr      equ $0D
0000: 33 esc      equ $1B
0000:
0000: 35 numbanks      equ $478-$C0      ;number of 64K banks on card
0000: 36 powerup      equ $4F8-$C0      ;powerup byte
0000: 37 power2      equ $578-$C0
0000:
0000: 39 * hardware equates, MUST be in $BF00 to avoid double access
0000:
0000: BFF8      equ $BFF8      ;address pointers
0000: BFF9      equ $BFF9      ;auto incs every data access
0000: BFFA      equ $BFFA      ;
0000: BFFB      equ $BFFB      ;data pointed to
0000:
0000: C000      equ $C000
0000: C010      equ $C010
0000: FC42      equ $FC42
0000: FC58      equ $FC58
0000: FC9C      equ $FC9C
0000: FD8E      equ $FD8E
0000: FD0A      equ $FD0A
0000: FD52      equ $FD52
0000: FD5D      equ $FD5D
```

```

2031:      2031      84 AddressTest equ *
2031:A9 05      lda #5
2033:85 25      sta CV
2035:20 8E FD      jsr Crout
2038:A9 10      lda #S10
203A:20 1D 22      jsr Print
203D:A5 4A      lda LoopCount+1
203F:20 DA FD      jsr PrByte
2042:A5 49      lda LoopCount
2044:20 DA FD      jsr PrByte
2047:20 0B 22      jsr WriteLine
204A:A9 01      lda #1
204C:85 00      sta TestNum
204E:A0 05      ldy #5
2050:B9 F4 22      lda Patterns,Y
2053:20 13 22      jsr serAddr
2056:DD F8 BF      cmp addr1,X
2059:DD 11 206C      bne atf
205B:DD F9 BF      cmp addrm,X
205E:DD 0C 206C      bne atf
2060:09 F0 104      ora #SF0
2062:DD FA BF      cmp addrh,X
2065:DD 05 206C      bne atf
2067:88      dey
2068:10 E6 2050      bpl at1
206A:30 03 206F      bmi RolloverTest
206C:4C 97 21      jmp Fail
206F:      206F      112 RolloverTest equ *
206F:E6 00      inc TestNum
2071:DE F8 BF      dec addr1,X
2074:BD F8 BF      lda data,X
2077:9D F8 BF      sta data,X
207A:BD FA BF      lda addrh,X
207D:29 0F      and #S0F
207F:1D F9 BF      ora addr1,X
2082:1D F8 BF      ora addrm,X
2085:F0 03 208A      beq AddBusTest
2087:4C 97 21      jmp Fail

```

```

----- NEXT OBJECT FILE NAME IS ASM.S.0
2000:      2000      org $2000
2000:      2000      MBR OFF
2000:      2000      *
2000:A9 00      lda #0
2002:85 49      sta LoopCount
2004:85 4A      sta LoopCount+1
2006:99 38 04      sta PowerUp,Y
2009:99 38 04      sta Power2,Y
200C:B3 B8 03      lda numbanks,Y
200F:29 0F      and #S0F
2011:85 46      sta Limit
2013:20 58 FC      jsr Home
2016:A9 08      lda #8
2018:      2018      * "MEMORY CARD TEST<CR>ESC TO EXIT<CR>TEST WILL TAKE "
2018:20 1D 22      jsr print
201B:A5 46      lda Limit
201D:4A      lsr A
201E:4A      lsr A
201F:48      pha
2020:09 04      ora #4
2022:20 1D 22      jsr print
2025:A9 09      lda #9
2027:20 1D 22      jsr print
202A:68      pla
202B:20 1D 22      jsr print
202E:20 8E FD      jsr Crout

```

02 S.DIAG1.SRC	slinky diagnostics	20-OCT-86 06:36 PAGE 5	02 S.DIAG1.SRC	slinky diagnostics	20-OCT-86 06:36 PAGE 6
208A: 124 *****	124 *****		20E3:10 E3 20C8 182	bpl ab4	
208A: 125 * Walk a 1 through the address registers to test for bus shorts	125 * Walk a 1 through the address registers to test for bus shorts		20E5:30 03 20EA 183	ClearTest	
208A: 126 * Walk a 1 through the address registers to test for bus shorts	126 * Walk a 1 through the address registers to test for bus shorts		20E7:4C 97 21 184	abFail jmp Fail	
208A: 127 * assumes addresses = 0 from previous test	127 * assumes addresses = 0 from previous test		20EA: 186 *****		
208A: 128 *	128 *		20EA: 187 *		
208A: 129 *****	129 *****		20EA: 188 *****		
208A: 130 AddrBusTest equ *	130 AddrBusTest equ *		20EA: 190 ClearTest equ *		
208A: 131 inc TestNum	131 inc TestNum		20EA: 20 11 22 191	jsr claddr	
208A: 132 lda #1	132 lda #1		20ED: 20ED 192	FillTest equ *	
208A: 133 sta CompData	133 sta CompData		20ED: 86 00 193	inc TestNum	
208A: 134 txa	134 txa		20ED: 85 45 194	sta CompData	
208A: 135 clc	135 clc		20F1:A5 45 195	lda CompData	
208A: 136 adc	136 adc		20F3:90 FB BF 196	sta Data,X	
208A: 137 lda #5C0	137 lda #5C0		20F6:90 FB BF 197	sta Data,X	
208A: 138 lda #5C0	138 lda #5C0		20F9:90 FB BF 198	sta Data,X	
208A: 139 sta MPtr+1	139 sta MPtr+1		20FC:90 FB BF 199	sta Data,X	
208A: 140 lda limit	140 lda limit		20FF:BD FB BF 200	lda addr1,X	
208A: 141 beq ab1	141 beq ab1		2102:D0 ED 20F1 201	bne f1	
208A: 142 cmp #50C	142 cmp #50C		2104:1D F9 BF 202	ora addrm,X	
208A: 143 bne ab2	143 bne ab2		2107:D0 E8 20F1 203	bne f1	
208A: 144 ab1	144 ab1		2109:20 E4 21 204	jsr PrBot	
208A: 145 ab2	145 ab2		210C:D0 E3 20F1 205	bne f1	
208A: 146 pha	146 pha		210E:20 05 22 206	jsr MtxLine	
208A: 147 ldy	147 ldy		2111:BD FB BF 207	lda Data,X	
208A: 148 ab3	148 ab3		2114:C5 45 208	cmp CompData	
208A: 149 jsr claddr	149 jsr claddr		2116:D0 CF 20E7 209	bne abFail	
208A: 150 pla	150 pla		2118:BD FB BF 210	lda Data,X	
208A: 151 sta (MPtr),y	151 sta (MPtr),y		211B:C5 45 211	cmp CompData	
208A: 152 pha	152 pha		211D:D0 C8 20E7 212	bne abFail	
208A: 153 lda	153 lda		211F:BD FB BF 213	lda addr1,X	
208A: 154 sta data,x	154 sta data,x		2122:D0 ED 2111 214	bne cpl	
208A: 155 inc CompData	155 inc CompData		2124:1D F9 BF 215	ora addrm,X	
208A: 156 pla	156 pla		2127:D0 E8 2111 216	bne cpl	
208A: 157 isr A	157 isr A		2129:20 E4 21 217	jsr PrBot	
208A: 158 bne ab3	158 bne ab3		212C:D0 E3 2111 218	bne cpl	
208A: 159 sta (MPtr),y	159 sta (MPtr),y		212E:20 05 22 219	jsr MtxLine	
208A: 160 ror A	160 ror A		2131:A5 45 220	lda CompData	
208A: 161 dey	161 dey		2133:49 FF 221	eor #FF	
208A: 162 bpl ab3	162 bpl ab3		2135:D0 B6 20ED 222	bne FillTest	
208A: 163 lda #1	163 lda #1				
208A: 164 sta CompData	164 sta CompData				
208A: 165 pla	165 pla				
208A: 166 ldy	166 ldy				
208A: 167 pha	167 pha				
208A: 168 jsr claddr	168 jsr claddr				
208A: 169 pla	169 pla				
208A: 170 sta	170 sta				
208A: 171 sta Value	171 sta Value				
208A: 172 lda data,x	172 lda data,x				
208A: 173 cmp CompData	173 cmp CompData				
208A: 174 bne abFail	174 bne abFail				
208A: 175 inc CompData	175 inc CompData				
208A: 176 lda Value	176 lda Value				
208A: 177 isr A	177 isr A				
208A: 178 bne ab4	178 bne ab4				
208A: 179 sta (MPtr),y	179 sta (MPtr),y				
208A: 180 ror A	180 ror A				
208A: 181 dey	181 dey				

02 S.DIAG1.SRC

slinky diagnostics

```

2137:      2137 224 Computed equ *
2137:R6 00      225 inc TestNum
2139:R9 55      226 lda #555
2139:R5 45      227 sta CompData
213D:20 FD 21  228 c1  getvalue
2140:18      229 c2  clc
2141:65 47      230 adc Value
2143:65 45      231 adc CompData
2145:8D F8 BF  232 sta data,x
2148:85 45      233 sta CompData
214A:8D F8 BF  234 lda addr1,x
214D:00 F1 2140 235 bne c2
214F:8D F9 BF  236 lda addr1,x
2152:00 E9 213D 237 bne c1
2154:20 E4 21  238 jsr PrDot
2157:00 E4 213D 239 bne c1
2159:20 08 22  240 jsr NxtLine
215C:A9 55 241 lda #555
215F:85 45      242 sta CompData
2160:20 FD 21  243 c3  jsr getvalue
2163:18      244 c4  clc
2164:65 47      245 Value
2166:65 45      246 adc CompData
2168:85 45      247 sta data,x
216A:8D F8 BF  248 lda CompData
216D:C5 45      249 cmp Fail
216F:00 26 2197 250 bne c4
2171:8D F8 BF  251 lda addr1,x
2174:00 ED 2163 252 bne c4
2176:8D F9 BF  253 lda addr1,x
2179:00 E5 2160 254 bne c3
217B:20 E4 21  255 jsr PrDot
217E:00 E0 2160 256 bne c3

```

;each byte gets computed value

;Test: 6

;Starting data pattern

;Address left at 0 from last test

;Value = addr1 + addrh + \$55, A = 0

;Save for next add

;Time to print a dot?

;Z = 1 if done

;Go to next line and clear address

;Starting data pattern

;Now read em back

;Is it right?

;Time to print a dot?

;Z = 1 if done

02 S.DIAG1.SRC

slinky diagnostics

```

2180:      2180 258 Pass equ *
2180:A9 08      259 lda #503
2182:20 1D 22  260 jsr Print
2183:F8 sed      261 LoopCount
2186:A5 49      262 lda #1
2188:18      263 clc
2189:59 01      264 adc LoopCount
218B:85 49      265 sta LoopCount+1
218D:A5 4A      266 lda LoopCount+1
218F:69 00      267 adc #0
2191:85 4A      268 sta LoopCount+1
2193:08      269 cld
2194:4C 31 20  270 jmp AddressTest
2197:      2197 271 *
2197:      2197 272 Fail equ *
2197:48      273 pha
2198:20 42 FC  274 jsr ClrEop
219B:A9 0A      275 lda #50A
219D:20 1D 22  276 jsr Print
21A0:A5 00      277 lda TestNum
21A2:C9 03      278 cmp #3
21A5:80 09 21AF 279 bcs DataErr
21A6:68      280 pla
21A7:A9 0C      281 lda #50C
21A9:20 1D 22  282 jsr Print
21AC:4C DE 21  283 jmp ErrCommon
21AF:A9 00      284 lda #50D
21B1:20 1D 22  285 jsr Print
21B4:38      286 sec
21B5:8D F8 BF  287 lda addr1,x
21B8:29 01      288 sbc #1
21BA:48      289 pha
21BB:8D F9 BF  290 lda addr1,x
21BE:E9 00      291 sbc #0
21C0:48      292 pla
21C1:8D FA BF  293 lda addr1,x
21C4:29 0F      294 and #50F
21C6:E9 00      295 sbc #0
21C8:20 DA FD  296 jsr PrByte
21CB:68      297 pla
21CC:20 DA FD  298 jsr PrByte
21CF:68      299 pla
21D0:20 DA FD  300 jsr PrByte
21D3:A9 0E      301 lda #50E
21D5:20 1D 22  302 jsr Print
21D8:68      303 pla
21D9:45 45      304 eor CompData
21DB:20 DA FD  305 jsr PrByte
21DE:A9 0F      306 ErrCommon
21E0:20 1D 22  307 lda #50F
21E3:60      308 jsr Print
21E4:      309 rts
21E4:      310 *
21E4:      311 *
21E4:      21E4 312 PrDot equ *
21E4:AN AE      313 lda #Dot
21E6:20 ED FD  314 jsr Count
21E9:AD 00 C0  315 lda Rdd

```

;passed all the tests

;CARD OK"

LoopCount

#1

LoopCount

LoopCount+1

LoopCount+1

AddressTest

;display failure message

;save actual data

;"CARD FAILED!&lt;CR&gt;"

;not an addressing problem

;there is no failing data really

;"ADDRESS ERROR"

;"DATA ERROR "

;set back to actual failing value

;propagate borrows (if any)

;mask off high 4 bits

;print as two hex digits

;print addr as two hex digits

;print addr1 as two hex digits

;" - "

;actual data

;print failing data as two hex digits

;"&lt;CR&gt;SEE DEALER FOR SERVICE&lt;CR&gt;"

;Is escape pressed?

[illegible]

00 CR  
 45 COMDATA  
 AE DOT  
 ?2000 STARTTEST  
 206F ROLLOVERTEST  
 20A8 AB3  
 20A8 AB4  
 20F1 F1  
 206D FILITEST  
 213D C1  
 2140 C2  
 ?2180 PASS  
 2184 PROOT  
 21F5 NOESC  
 2213 SETADDR  
 2241 M0  
 2230 MESSAGES  
 224E M3  
 2252 M4  
 225C M8  
 2259 M7  
 22AE M0B  
 2288 M0C  
 22EB M10  
 BFF8 ADDR  
 BFF9 ADDR  
 C010 KSTROBE  
 FC9C CLRROL  
 F08E CROUT

07 BELL  
 42 MPTR  
 49 LOOPCOUNT  
 0488 POWER2  
 206C ATF  
 20A4 AB2  
 20A8 AB2  
 206D FILITEST  
 ?2137 COMPUTED  
 2111 CP1  
 2160 C3  
 21A6 DATAERR  
 21F0 GETVALUE  
 2208 NYTLIN  
 2222 P11  
 224A M2  
 2251 M6  
 2290 M0A  
 22D0 M0E  
 ?233C ZSID.END  
 BFF8 DATA  
 C000 KBD  
 FC9C CLRROL  
 F08E CROUT  
 F0DA PRBYTE  
 F0ED COUT

\*\* SUCCESSFUL ASSEMBLY := NO ERRORS  
 \*\* ASSEMBLER CREATED ON 15-JAN-84 21:28  
 \*\* TOTAL LINES ASSEMBLED 417  
 \*\* FREE SPACE PAGE COUNT 82

20A4 AB2  
 208A ADDRESSTEST  
 BFF9 ADDR  
 213D C1  
 20EA CLEARTEST  
 45 COMDATA  
 F08E CROUT  
 21A6 DATAERR  
 20F1 F1  
 FC58 HOME  
 49 LOOPCOUNT  
 22D0 M0E  
 2246 M1  
 2252 M4  
 225C M8  
 21F5 NOESC  
 22F4 PATTERNS  
 F0DA PRBYTE  
 2213 SETADDR  
 ?233C ZSID.END  
 20A8 AB3  
 2031 ADDRESSTEST  
 2050 AT1  
 2140 C2  
 2211 CLRADDR  
 ?2137 COMPUTED  
 00 CR  
 AE DOT  
 2197 FAIL  
 C000 KBD  
 2290 M0A  
 2241 M0  
 22EB M10  
 225B M5  
 2257 M6  
 2230 MESSAGES  
 0388 NUMBANKS  
 0488 POWER2  
 21E4 PROOT  
 ?2000 STARTTEST  
 206F ROLLOVERTEST  
 47 VALUE  
 20A4 AB4  
 BFFA ADDR  
 206C ATF  
 2160 C3  
 FC9C CLRROL  
 F0ED COUT  
 25 CV  
 21DE ERRCOMMON  
 206D FILITEST  
 C010 KSTROBE  
 22AE M0B  
 21C5 M0D  
 224A M2  
 2255 M5  
 2288 M9  
 22C5 M0D  
 22F4 PATTERNS  
 BFFA ADDR  
 FC42 CLRROL  
 F0DA PRBYTE  
 F0ED COUT  
 00 TESTNUM





# Glossary

**accumulator:** The register in the 65C02 microprocessor where most computations are performed.

**ACIA:** Acronym for *Asynchronous Communications Interface Adapter*. A single **chip** that converts data from parallel to serial form and vice versa. An ACIA handles serial transmission and reception and RS-232-C signals under the control of its internal registers, which can be set and changed by **firmware** or **software**.

**acronym:** A word formed from the initial letters of a name or phrase, such as ROM (from **read-only memory**).

**ADC:** See **analog-to-digital converter**.

**address:** A number that specifies the location of a single **byte** of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from \$0000 to \$FFFF (in hexadecimal).

**algorithm:** A step-by-step procedure for solving a problem or accomplishing a task.

**American Simplified Keyboard:** See **Dvorak keyboard**.

**analog:** Varying smoothly and continuously over a range, rather than changing in discrete jumps. For example, a conventional 12-hour clock face is an analog device that shows the time of day by the continuously changing position of the clock's hands. Compare **digital**.

**analog data:** Data in the form of continuously variable quantities. Compare **digital data**.

**analog signal:** A signal that varies continuously over time, rather than being sent and received in discrete intervals. Compare **digital signal**.

**analog-to-digital converter (ADC):** A device that converts quantities from analog to digital form. For example, computer hand controls convert the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes stepwise even when the dial is turned smoothly.

**AND:** A logical operator that produces a true result if both its operands are true, and a false result if either or both its operands are false. Compare **OR**, **NOT**, **exclusive OR**.

**ANSI:** Acronym for *American National Standards Institute*, which sets standards for many technical fields and is the most common standard for computer terminals.

**Apple I:** The first Apple computer. It was built in a garage in California by Steve Jobs and Steve Wozniak.

**Applesoft BASIC:** The Apple II dialect of the BASIC programming language. An interpreter for creating and executing Applesoft BASIC programs is built into the firmware of computers in the Apple II family. See also **BASIC**, **Integer BASIC**.

**Apple III:** An Apple computer; part of the Apple II family. The Apple III offered a built-in disk drive and built-in RS-232-C (serial) port. Its memory was expandable to 256K.

**Apple II:** A family of computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS. The original Apple II used Integer BASIC instead of Applesoft BASIC, and it required a keyboard command (PR#6) in order to start up from a disk.

**Apple IIc:** A transportable personal computer in the Apple II family, with a disk drive and 80-column display capability built in.

**Apple IIe:** A personal computer in the Apple II family with seven expansion slots and an auxiliary memory slot that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards. The Apple IIe has been improved and enhanced over the years.

**Apple IIe 80-Column Text Card:** A peripheral card that plugs into the Apple IIe's auxiliary memory slot and allows the computer to display either 40 or 80 characters per line.

**Apple IIe Extended 80-Column Text Card:** A peripheral card that plugs into the Apple IIe's auxiliary memory slot and allows the computer to display either 40 or 80 characters per line while extending the computer's memory capacity by 64K.

**Apple IIGS:** A powerful new member of the Apple II family. The Apple IIGS uses a 16-bit microprocessor and has 256K of RAM. It has slots like the Apple IIe and ports like the Apple IIc, and contains a 15-voice custom sound chip.

**Apple II Pascal:** A software system for the Apple II family that lets you create and execute programs written in the Pascal programming language. Apple II Pascal was adapted by Apple Computer from the University of California, San Diego, Pascal Operating System (UCSD Pascal).

**Apple II Plus:** A personal computer in the Apple II family with expansion slots that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards.

**application program:** A program written for some specific purpose, such as word processing, data base management, graphics, or telecommunication. Compare **system program**.

**argument:** A value on which a function or statement operates; it can be a number or a variable. For example, in the BASIC statement VTAB 10, the number 10 is the argument. Compare **operand**.

**arithmetic expression:** A combination of numbers and arithmetic operators (such as 3 + 5) that indicates some operation to be carried out.

**arithmetic operator:** An operator, such as +, that combines numeric values to produce a numeric result. Compare **logical operator**, **relational operator**.

**ASCII:** Acronym for *American Standard Code for Information Interchange*; pronounced "ASK-ee." A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device. Compare **EBCDIC**.

**assembler:** A language translator that converts a program written in assembly language into an equivalent program in machine language. The opposite of a **disassembler**.

**assembly language:** A low-level programming language in which individual machine-language instructions are written in a symbolic form that's easier to understand than machine language itself. Each assembly-language instruction produces one machine-language instruction. See also **machine language**.

**asynchronous:** Not synchronized by a mutual timing signal or clock. Compare **synchronous**.

**asynchronous transmission:** A method of data transmission in which the receiving and sending devices don't share a common timer, and no timing data is transmitted. Each information character is individually synchronized, usually by the use of start and stop bits. The time interval between characters isn't necessarily fixed. Compare **synchronous transmission**.

**auxiliary slot:** The special expansion slot inside the Apple IIe used for the Apple IIe 80-Column Text Card or Extended 80-Column Text Card, and also for the **RGB monitor** card. The slot is labeled AUX. CONNECTOR on the circuit board.

**back panel:** The rear surface of the computer, which includes the power switch, the power connector, and connectors for peripheral devices.

**bandwidth:** The range of frequencies a device can handle. Bandwidth and maximum data transfer rate are directly proportional. For example, a video monitor's greater bandwidth allows it to display more information per scan frame than most home television sets can. To display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

**base address:** In *indexed addressing*, the fixed component of an address.

**BASIC:** Acronym for *Beginners All-purpose Symbolic Instruction Code*. BASIC is a high-level programming language designed to be easy to learn. Two versions of BASIC are available from Apple Computer for use with all Apple II-family systems: Applesoft BASIC (built into the firmware) and Integer BASIC.

**baud:** A unit of data transmission speed: the number of discrete signal state changes per second. Often, but not always, equivalent to *bits per second*. Compare **bit rate**.

**binary:** Characterized by having two different components, or by having only two alternatives or values available; sometimes used synonymously with **binary system**.

**binary digit:** The smallest unit of information in the binary number system; a 0 or a 1. Also called a **bit**.

**binary operator:** An operator that combines two operands to produce a result. For example, + is a binary arithmetic operator; < is a binary relational operator; OR is a binary logical operator. Compare **unary operator**.

**binary system:** The representation of numbers in the base-2 system, using only the two digits 0 and 1. For example, the numbers 0, 1, 2, 3, and 4 become 0, 1, 10, 11, and 100 in binary notation. The binary system is commonly used in computers because the values 0 and 1 can easily be represented in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen. A single binary digit—a 0 or a 1—is called a **bit**. Compare **decimal**, **hexadecimal**.

**bit:** A contraction of *binary digit*. The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing. See also **binary system**.

**bit rate:** The speed at which bits are transmitted, usually expressed as *bits per second*, or *bps*. Compare **baud**.

**bits per second:** See **bit rate**.

**board:** See **printed-circuit board**.

**body:** In BASIC, the statements or instructions that make up a part of a program, such as a loop or a subroutine.

**boot:** Another way to say **start up**. A computer boots by loading a program into memory from an external storage medium such as a disk. Starting up is often accomplished by first loading a small program, which then reads a larger program into memory. The program is said to “pull itself up by its own bootstraps”—hence the term *bootstrapping* or *booting*.

**boot disk:** See **startup disk**.

**bootstrap:** See **boot**.

**bps:** See **bit rate**.

**branch:** (v) To pass program control to a line or statement other than the next in sequence. (n) A statement that performs a branch. See **conditional branch**, **unconditional branch**.

**BREAK:** A SPACE (0) signal, sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a time-sharing service. BREAK is also used in BASIC to stop execution of a program; it's generated by pressing Control-C.

**BRK:** A “software interrupt.” An instruction that causes the 6502 or 65C02 microprocessor to halt. Pronounced “break.”

**buffer:** A “holding area” of the computer's memory where information can be stored by one program or device and then read at a different rate by another; for example, a print buffer. In editing functions, an area in memory where deleted (cut) or copied data is held. In some applications, this area is called the *Clipboard*.

**bug:** An error in a program that causes it not to work as intended. The expression reportedly comes from the early days of computing when an itinerant moth shorted a connection and caused a breakdown in a room-size computer.

**bus:** A group of wires or circuits that transmit related information from one part of a computer system to another. In a network, a line of cable with connectors linking devices together. A bus network has a beginning and an end. (It's not in a closed circle or T shape.)

**byte:** A unit of information consisting of a fixed number of **bits**. On Apple II systems, one byte consists of a series of eight bits, and a byte can represent any value between 0 and 255. The sequence represents an instruction, letter, number, punctuation mark, or other character. See also **kilobyte**, **megabyte**.

**cable:** An insulated bundle of wires with connectors on the ends; the number of wires varies with the type of connection. Examples are serial cables, disk drive cables, and AppleTalk cables.

**call:** (v) To request the execution of a subroutine, function, or procedure. (n) A request from the keyboard or from a procedure to execute a named procedure. See **procedure**.

**carriage return:** An ASCII character (decimal 13) that ordinarily causes a printer or display device to place the next character on the left margin.

**carrier:** The background signal on a communication channel that is modified to carry information. Under RS-232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

**carry flag:** A status bit in the 6502 or 65C02 microprocessor, used as a ninth bit with the eight accumulator bits in addition, subtraction, rotation, and shift operations.

**cathode-ray tube (CRT):** An electronic device, such as a television picture tube, that produces images on a phosphor-coated screen. The phosphor coating emits light when struck by a focused beam of electrons. A common display device used with personal computers.

**central processing unit (CPU):** The “brain” of the computer; the microprocessor that performs the actual computations in machine language. See **microprocessor**.

**character:** Any symbol that has a widely understood meaning and thus can convey information. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer. Compare **control character**.

**character code:** A number used to represent a character for processing by a computer system.

**character set:** The entire set of characters that can be either shown on a monitor or used to code computer instructions. In a printer, the entire set of characters that the printer is capable of printing.

**chip:** See **integrated circuit**.

**Clear To Send:** An RS-232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out. See **Data Communication Equipment**, **Data Terminal Equipment**.

**code:** (1) A number or symbol used to represent some piece of information. (2) The statements or instructions that make up a program.

**cold start:** The process of starting up the Apple II when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, and then loading and running a program. Compare **warm start**.

**column:** A vertical arrangement of graphics points or character positions on the display.

**command:** An instruction that causes the computer to perform some action. A command can be typed from a keyboard, selected from a menu with a hand-held device (such as a mouse), or embedded in a program.

**command character:** An ASCII character, usually Control-A or Control-I, that causes the serial port firmware to interpret subsequent characters as commands.

**command register:** An ACIA location (at \$C09A for port 1 and \$C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

**compiler:** A language translator that converts a program written in a high-level programming language (source code) into an equivalent program in some lower-level language such as machine language (object code) for later execution. Compare **interpreter**.

**composite video:** A video signal that includes both display information and the synchronization (and other) signals needed to display it. See **RGB monitor**.

**computer:** An electronic device that performs predefined (programmed) computations at high speed and with great accuracy. A machine that is used to store, transfer, and transform information.

**computer language:** See **programming language**.

**computer system:** A computer and its associated hardware, firmware, and software.

**conditional branch:** A **branch** whose execution depends on the truth of a condition or the value of an expression. Compare **unconditional branch**.

**configuration:** (1) The total combination of hardware components—CPU, video display device, keyboard, and peripheral devices—that make up a computer system. (2) The software settings that allow various hardware components of a computer system to communicate with each other.

**connector:** A plug, socket, jack, or port.

**constant:** In a program, a symbol that represents a fixed, unchanging value. Compare **variable**.

**control character:** A nonprinting character that controls or modifies the way information is printed or displayed. In the Apple II family, control characters have ASCII values between 0 and 31, and are typed from a keyboard by holding down the Control key while pressing some other key. In the Macintosh family, the Command key performs a similar function.

**control code:** One or more nonprinting characters—included in a text file—whose function is to change the way a printer prints the text. For example, a program may use certain control codes to turn boldface printing on and off. See **control character**.

**control key:** A general term for a key that controls the operation of other keys; for example, Apple, Caps Lock, Control, Option, and Shift. When you hold down or engage a control key while pressing another key, the combination makes that other key behave differently. Also called a *modifier key*.

**Control key:** A specific key on Apple II-family keyboards that produces **control characters** when used in combination with other keys.

**controller card:** A peripheral card that connects a device such as a printer or disk drive to a computer's main logic board and controls the operation of the device.

**control register:** An ACIA location ( at \$C09B for port 1 and \$C0AB for port 2) that stores data format and baud rate selections.

**Control-Reset:** A combination keystroke on Apple II-family computers that usually causes an Applesoft BASIC program or command to stop immediately. If a program disables the Control-Reset feature, you need to turn the computer off to get the program to stop.

**copy protect:** To make a disk uncopyable. Software publishers frequently try to copy protect their disks to prevent them from being illegally duplicated by software pirates. Compare **write protect**.

**CPU:** See **central processing unit**.

**CRT:** See **cathode-ray tube**.

**CTS:** See **Clear To Send**.

**crash:** To cease to operate unexpectedly, possibly destroying information in the process.

**current input device:** The source, such as the keyboard or a modem, from which a program is currently receiving its input.

**current output device:** The destination, such as the display screen or a printer, currently receiving a program's output.

**cursor:** A symbol displayed on the screen marking where the user's next action will take effect or where the next character typed from the keyboard will appear.

**DAC:** See **digital-to-analog converter**.

**data:** Information, especially information used or operated on by a program. The smallest unit of information a computer can understand is a **bit**.

**data bits:** The bits in a communication transfer that contain information. Compare **start bit**, **stop bit**.

**Data Carrier Detect (DCD):** An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIe) indicating that a communication connection has been established. See **Data Communication Equipment**, **Data Terminal Equipment**.

**Data Communication Equipment (DCE):** As defined by the RS-232-C standard, any device that transmits or receives information. Usually this device is a **modem**.

**data format:** The form in which data is stored manipulated, or transferred.

**data set:** A device that modulates, demodulates, and controls signals transferred between business machines and communication facilities. A form of **modem**.

**Data Set Ready (DSR):** An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection. See **Data Communication Equipment, Data Terminal Equipment**.

**Data Terminal Equipment (DTE):** As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as an endpoint of a communication connection. A computer might serve as a DTE.

**Data Terminal Ready (DTR):** An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data. See **Data Communication Equipment, Data Terminal Equipment**.

**DCD:** See **Data Carrier Detect**.

**DCE:** See **Data Communication Equipment**.

**debug:** A colloquial term that means to locate and correct an error or the cause of a problem or malfunction in a computer program. Compare **troubleshoot**. See also **bug**.

**decimal:** The common form of number representation used in everyday life, in which numbers are expressed in the base-10 system, using the ten digits 0 through 9. Compare **binary, hexadecimal**.

**default:** A preset response to a question or prompt. The default is automatically used by the computer if you don't supply a different response. Default values prevent a program from stalling or crashing if no value is supplied by the user.

**deferred execution:** The execution of a BASIC program instruction that is part of a complete program. The program instruction is executed only when the complete program is run. You defer execution of the instruction by preceding it with a program line number. The complete program executes consecutive instructions in numerical order. Compare **immediate execution**.

**Delete key:** A key on the upper-right corner of the Apple IIe and IIc keyboards that erases the character immediately preceding (to the left of) the cursor. Similar to the Macintosh Backspace key.

**delimiter:** A character that is used for punctuation to mark the beginning or end of a sequence of characters, and which therefore is not considered part of the sequence itself. For example, Applesoft BASIC uses the double quotation mark (") as a delimiter for string constants: the string "DOG" consists of the three characters *D*, *O*, and *G*, and does not include the quotation marks.

**demodulate:** To recover the information being transmitted by a modulated signal. For example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into the sound emitted by the radio's speaker. Compare **modulate**.

**device:** Frequently used as a short form of **peripheral device**.

**device driver:** A program that manages the transfer of information between the computer and a peripheral device.

**device handler:** See **device driver**.

**digit:** (1) One of the characters 0 through 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 through 9 and A through F in hexadecimal.

**digital:** Represented in a discrete (noncontinuous) form, such as numerical digits or integers. For example, contemporary digital clocks show the time as a digital display (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog**.

**digital data:** Data that can be represented by digits—that is, data that are discrete rather than continuously variable. Compare **analog data**.

**digital signal:** A signal that is sent and received in discrete intervals. A signal that does not vary continuously over time. Compare **analog signal**.

**digital-to-analog converter:** A device that converts quantities from digital to analog form.

**DIP:** See **dual in-line package**.

**DIP switches:** A bank of tiny switches, each of which can be moved manually one way or the other to represent one of two values (usually on and off). See **dual in-line package**.

**disassembler:** A language translator that converts a machine-language program into an equivalent program in assembly language, which is easier for programmers to understand. The opposite of an **assembler**.

**disk:** An information-storage medium consisting of a flat, circular, magnetic surface on which information can be recorded in the form of small magnetized spots, in a manner similar to the way sounds are recorded on tape. See **floppy disk**, **hard disk**.

**disk-based:** See **disk-resident**.

**disk controller card:** A peripheral card that provides the connection between one or two disk drives and the computer. This connection, or interface, is built into both the Apple IIc and Macintosh-family computers.

**disk drive:** The device that holds a disk, retrieves information from it, and saves information to it.

**disk envelope:** A removable, protective paper sleeve used when handling or storing a 5.25-inch disk. It must be removed before you insert the disk in a disk drive. Compare **disk jacket**.

**disk jacket:** A permanent, protective covering for a disk. 5.25-inch disks have flexible, paper or plastic jackets; 3.5-inch disks have hard plastic jackets. The disk is never removed from the jacket. Compare **disk envelope**.

**Disk Operating System (DOS):** An optional software system for the Apple II family of computers that enables the computer to control and communicate with one or more disk drives. The acronym *DOS* rhymes with *boss*.

**disk-resident:** An adjective describing a program that does not remain in memory. The computer retrieves all or part of the program from the disk, as needed. Sometimes called *disk-based*. Compare **memory-resident**.

**Disk II drive:** An older type of disk drive made and sold by Apple Computer for use with the Apple II, II Plus, and IIe. It uses 5.25-inch disks.

**display:** (1) A general term to describe what you see on the screen of your display device when you're using a computer; from the verb form, which means "to place into view." (2) Short for a display device.

**display color:** The color currently being used to draw high-resolution or low-resolution graphics on the display screen.

**display device:** A device that displays information, such as a television set or video monitor.

**display screen:** The screen of the monitor; the area where you view text and pictures when using the computer.

**DOS 3.2:** An early Apple II operating system. DOS stands for **Disk Operating System**; 3.2 is the version number. Disks formatted using DOS 3.2 have 13 sectors per track.

**DOS 3.3:** An operating system used by the Apple II family of computers. DOS stands for **Disk Operating System**; 3.3 is the version number. Disks formatted with DOS 3.3 have 16 sectors per track.

**drive:** See **disk drive**.

**DSR:** See **Data Set Ready**.

**DTE:** See **Data Terminal Equipment**.

**DTR:** See **Data Terminal Ready**.

**dual in-line package (DIP):** An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side. **DIP switches** on the box allow you to change settings. For example, ImageWriter printer DIP switches control functions such as line feed, form length, and **baud** setting.

**Dvorak keyboard:** An alternate keyboard layout, also known as the *American Simplified Keyboard*, which increases typing speed because the keys most often used are in the positions easiest to reach. Compare **QWERTY keyboard**.

**EBCDIC:** Acronym for *Extended Binary-Coded Decimal Interchange Code*; pronounced "EB-si-dik." A code used by IBM that represents each letter, number, special character, and control character as an 8-bit binary number. EBCDIC has a character set of 256 8-bit characters. Compare **ASCII**.

**effective address:** In machine-language programming, the address of the memory location on which a particular instruction operates, which may be arrived at by indexed addressing or some other addressing method.

**80-column text card:** A peripheral card that allows the Apple II, Apple II Plus, and Apple IIe to display text in either 40 columns or 80 columns.

**80/40-column switch:** A switch that controls the maximum number of columns or characters across the screen. A television can legibly display a maximum of 40 characters across the screen, whereas a video monitor can display 80 characters.

**embedded:** Contained within. For example, the string 'HUMPTY DUMPTY' is said to contain an embedded space.

**emulate:** To operate in a way identical to a different system. For example, the Apple II 2780/3780 Protocol Emulator and the Apple II 3270 BSC Protocol Emulator, together with the Apple Communications Protocol Card (ACPC), allow the Apple II, Apple II Plus, or Apple IIe to emulate the operations of IBM 3278 and 3277 terminals and 3274 and 3271 control units.

**emulation mode:** A mode of operation in which the computer is emulating the operation of another computer or interface. See **emulate**.

**end-of-command mark:** A punctuation mark used to separate commands sent to a peripheral device such as a printer or plotter. Also called a *command terminator*.

**end-of-line character:** A character which indicates that the preceding text constitutes a full line.

**error code:** A number or other symbol representing a type of error.

**error message:** A message displayed or printed to tell you of an error or problem in the execution of a program or in your communication with the system. An error message is often accompanied by a beep.

**escape character:** An ASCII character that, with many programs and devices, allows you to perform special functions when used in combination keypresses.

**escape code:** A sequence of characters that begins with an ESCAPE character and constitutes a complete command. Usually synonymous with **escape sequence**.

**Escape key:** A key on Apple II-family computers that generates the ESCAPE character. The Escape key is labeled *Esc*. In many applications, pressing Esc allows you to return to a previous **menu** or to stop a procedure.

**escape mode:** A state of the Apple IIe and IIc entered by pressing the Esc key and certain other keys. The other keys take on special meanings for positioning the cursor and controlling the display of text on the screen.

**escape sequence:** A sequence of keystrokes, beginning with the Esc key. In **escape mode**, escape sequences are used for positioning the cursor and controlling the display of text on the screen. Escape sequences are also used as codes to control printers.

**Esc key:** See **Escape key**.

**even/odd parity check:** In data transmission, a check that tests whether the number of 1 bits in a group of binary digits is even (even parity check) or odd (odd parity check).

**even parity:** In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an even number; used as a means of error checking. Compare **MARK parity**, **odd parity**.

**exclusive OR:** A logical operator that produces a true result if one of its operands is true and the other false, and a false result if its operands are both true or both false. Compare **OR**, **AND**, and **NOT**.

**execute:** To perform the actions specified by a program command or sequence of commands.

**expansion slot:** A connector into which you can install a peripheral card. Sometimes called a *peripheral slot*. See also **auxiliary slot**.

**expression:** A formula in a program that defines a calculation to be performed.

**FIFO:** Acronym for “first in, first out” order, as in a **queue**.

**file:** Any named, ordered collection of information stored on a disk. Application programs and operating systems on disks are files. You make a file when you create text or graphics, give the material a name, and save it to disk.

**firmware:** Programs stored permanently in read-only memory (ROM). Such programs (for example, the Applesoft Interpreter and the Monitor program) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory. Compare **hardware**, **software**.

**fixed-point:** A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number so that the number is interpreted as an **integer**. Compare **floating-point**.

**flag:** A variable whose value (usually 1 or 0, standing for *true* or *false*) indicates whether some condition holds or whether some event has occurred. A flag is used to control the program's actions at some later time.

**floating-point:** A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to “float” to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point**.

**floppy disk:** A **disk** made of flexible plastic, as compared to a **hard disk**, which is made of metal. The term *floppy* is now usually applied only to disks with thin, flexible **disk jackets**, such as 5.25-inch disks. With 3.5-inch disks, the disk itself is flexible, but the jacket is made of hard plastic; thus, 3.5-inch disks aren't particularly “floppy.”

**format:** (n) (1) The form in which information is organized or presented. (2) The general shape and appearance of a printed page, including page size, character width and spacing, line spacing, and so on. (v) To divide a disk into tracks and sectors where information can be stored. Blank disks must be formatted before you can save information on them for the first time; same as **initialize**.

**form feed:** An ASCII character (decimal 12) that causes a printer or other paper-handling device to advance to the top of the next page.

**Fortran:** Short for *Formula Translator*. A high-level programming language especially suitable for applications requiring extensive numerical calculations, such as in mathematics, engineering, and the sciences.

**framing error:** In serial data transfer, the absence of the expected stop bit(s) at the end of a received character.

**frequency:** In alternating current (AC) signals, the number of complete cycles transmitted per second. Frequency is usually expressed in hertz (cycles per second), kilohertz (kilocycles per second), or megahertz (megacycles per second). In acoustics, frequency of vibration determines musical pitch. Compare **duration**.

**full duplex:** A four-wire communication circuit or protocol that allows two-way data transmission between two points at the same time. Compare **half duplex**.

**function:** A preprogrammed calculation that can be carried out on request from any point in a program. A function takes in one or more arguments and returns a single value. It can therefore be embedded in an expression.

**game I/O connector:** A 16-pin connector inside the Apple II, II Plus, and IIe, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector**.

**graph:** A pictorial representation of data.

**graphics:** (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

**half duplex:** A two-wire communication circuit or protocol designed for data transmission in either direction but not both directions simultaneously. Compare **full duplex**.

**hand control connector:** A 9-pin connector on the back panel of the Apple IIe and IIc computers, used for connecting hand controls to the computer. Compare **game I/O connector**.

**hand controller:** Peripheral devices, with rotating dials and push buttons. Hand controllers are used to control game-playing programs, but they can also be used in other applications.

**hang:** To cease operation because either an expected condition is not satisfied or an infinite loop is occurring. A computer that's hanging is called a *hung system*. Compare **crash**.

**hard disk:** A disk made of metal and sealed into a drive or cartridge. A hard disk can store very large amounts of information compared to a **floppy disk**.

**hard disk drive:** A device that holds a hard disk, retrieves information from it, and saves information to it. Hard disks made for microprocessors are permanently sealed into the drives.

**hardware:** In computer terminology, the machinery that makes up a computer system. Compare **firmware**, **software**.

**hertz:** The unit of frequency of vibration or oscillation, defined as the number of *cycles per second*. Named for the physicist Heinrich Hertz and abbreviated *Hz*. The 6502 microprocessor used in the Apple II systems operates at a clock frequency of about 1 million hertz, or 1 megahertz (MHz). The 68000 microprocessor used in the Macintosh operates at 7.8336 MHz.

**hexadecimal:** The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. For example, the decimal numbers 0, 1, 2, 3, 4, ... 8, 9, 10, 11, ... 15, 16, 17 would be shown in hexadecimal notation as 00, 01, 02, 03, 04, ... 08, 09, 0A, 0B, ... 0F, 10, 11. Hexadecimal numbers are easier for people to read and understand than are binary numbers, and they can be converted easily and directly to binary form. Each hexadecimal digit corresponds to a sequence of four **binary digits**, or bits. Hexadecimal numbers are usually preceded by a dollar sign (\$).

**high ASCII characters:** ASCII characters with decimal values of 128 to 255. Called *high ASCII* because their high bit (first binary digit) is set to 1 (for *on*) rather than 0 (for *off*).

**high-level language:** A programming language that is relatively easy for people to understand. A single statement in a high-level language typically corresponds to several instructions of machine language. High-level languages available from Apple Computer include BASIC, Pascal, Instant Pascal, Logo, Pilot, SuperPILOT, and Fortran. Compare **low-level language**.

**high-order byte:** The more significant half of a memory address or other two-byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the **low-order byte** of an address is usually stored first, and the high-order byte second. In the 68000 microprocessors used in the Macintosh family, the high-order byte is stored first.

**high-resolution graphics:** The display of graphics on a screen as a six-color array of points, 280 columns wide and 192 rows high. When a text window is in use, the visible high-resolution graphics display is 280 by 160 points.

**hold time:** In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off. Compare **setup time**.

**Hz:** See **hertz**.

**IC:** See **integrated circuit**.

**immediate execution:** The execution of a program statement as soon as it is typed. In BASIC, immediate execution occurs when the line is typed without a line number; immediate execution allows you to try out nearly every statement immediately to see how it works. Compare **deferred execution**.

**implement:** To put into practical effect, as to *implement* a plan. For example, a language translator implements a particular language.

**IN#:** This command designates the source of subsequent input characters. It can be used to designate a device in a slot or a machine-language routine as the source of input.

**index:** (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

**indexed addressing:** A method used in machine language programming to specify memory addresses. See also **memory location**.

**index register:** A register in a computer processor that holds an index for use in indexed addressing. The 6502 microprocessor used in the Apple II family of computers has two index registers, called the **X register** and the **Y register**. The 68000 microprocessor used in Macintosh-family computers has 16 registers that can be used as index registers.

**index variable:** A variable whose value changes on each pass through a loop. Often called *control variable* or *loop variable*.

**infinite loop:** A section of a program that will repeat the same sequence of actions indefinitely.

**initialize:** (1) To set to an initial state or value in preparation for some computation. (2) To prepare a blank disk to receive information by organizing its surface into tracks and sectors; same as **format**.

**initialized disk:** A disk that has been organized into tracks and sectors by the computer and is therefore ready to store information.

**input:** Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem.

**input/output (I/O):** The process by which information is transferred between the computer's memory and its keyboard or peripheral devices.

**input routine:** A machine-language routine; the standard input routine reads characters from the keyboard. A different input routine might, for example, read them from an external terminal.

**instruction:** A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

**integer:** A whole number in fixed-point form. Compare **real number**.

**Integer BASIC:** A version of the BASIC programming language used by the Apple II family of computers. Integer BASIC is older than Applesoft BASIC and is capable of processing numbers in integer (fixed-point) form only. Many games are written in Integer BASIC because its instructions can be executed very quickly. Compare **Applesoft BASIC**.

**integrated circuit:** An electronic circuit—including components and interconnections—entirely contained in a single piece of semiconducting material, usually silicon. Often referred to as an *IC* or a *chip*.

**interface:** (1) The point at which independent systems or diverse groups interact. The devices, rules, or conventions by which one component of a system communicates with another. Also, the point of communication between a person and a computer. (2) The part of a program that defines constants, variables, and data structures, rather than procedures.

**interface card:** A peripheral card that implements a particular interface (such as a parallel or serial interface) by which the computer can communicate with a peripheral device such as a printer or modem.

**interpreter:** A language translator that reads a program instruction by instruction and immediately translates each instruction for the computer to carry out. Compare **compiler**.

**interrupt:** A temporary suspension in the execution of a program that allows the computer to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

**inverse video:** The display of text on the computer's display screen in the form of dark dots on a light (or other single phosphor color) background, instead of the usual light dots on a dark background.

**I/O:** See **input/output**.

**I/O device:** Input/output device. A device that transfers information into or out of a computer. See **input**, **output**, **peripheral device**.

**I/O link:** A fixed location that contains the address of an input/output subroutine in the computer's Monitor program.

**IWM:** "Integrated Woz Machine"; the custom chip that controls Apple's 3.5-inch disk drives.

**joystick:** A peripheral device with a lever, typically used to move creatures and objects in game programs; a joystick can also be used in applications such as computer-aided design and graphics programs.

**K:** See **kilobyte**.

**keyboard:** The set of keys, similar to a typewriter keyboard, used for entering information into the computer.

**keyboard input connector:** The connector inside the Apple II family of computers by which the keyboard is connected to the computer.

**keyword:** A special word or sequence of characters that identifies a particular type of statement or command, such as *RUN*, *BRUN*, or *PRINT*.

**kilobyte (K):** A unit of measurement consisting of 1024 ( $2^{10}$ ) **bytes**. In this usage, *kilo* (from the Greek, meaning a thousand) stands for 1024. Thus, 64K memory equals 65,536 bytes. See also **megabyte**.

**KSW:** The symbolic name of the location in the computer's memory where the standard input link (namely, to the keyboard) is stored. *KSW* stands for *keyboard switch*.

**language:** See **programming language**.

**language card:** A peripheral card that, when placed in slot 0 of a 48K Apple II or Apple II Plus, gives the computer a total of 64K of memory. If you have an Apple II or Apple II Plus, you need a language card or the equivalent to use ProDOS.

**language translator:** A system program that reads another program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See **interpreter**, **compiler**, **assembler**.

**leading zero:** A zero occurring at the beginning of a decimal number, deleted by most computing programs.

**least significant bit:** The rightmost bit of a binary number. The least significant bit contributes the smallest quantity to the value of the number. Compare **most significant bit**.

**LIFO:** Acronym for "last in, first out" order, as in a **stack**.

**line:** See **program line**.

**line feed:** An ASCII character (decimal 10) that ordinarily causes a printer or video display to advance to the next line.

**line number:** A number identifying a program line in an Applesoft BASIC program.

**line width:** The number of characters that fit on a line on the screen or on a page.

**list:** To display on a monitor, or print on a printer, the contents of memory or of a file.

**load:** To transfer information from a peripheral storage medium (such as a disk) into main memory for use—for example, to transfer a program into memory for execution.

**local:** Connected to or close by the host system.

**location:** See **memory location**.

**logic:** (1) In microcomputers, a mathematical treatment of formal logic using a set of symbols to represent quantities and relationships that can be translated into switching circuits, or *gates*. AND, OR, and NOT are examples of logical gates. Each gate has two states, open or closed, allowing the application of **binary** numbers for solving problems. (2) The systematic scheme that defines the interactions of signals in the design of an automatic data processing system.

**logical operator:** An operator, such as AND, that combines logical values to produce a logical result, such as true or false; sometimes called a *Boolean operator*. Compare **arithmetic operator**, **relational operator**.

**logic board:** See **main logic board**.

**loop:** A section of a program that is executed repeatedly until a limit or condition is met, such as an index variable's reaching a specified ending value. See **loop**.

**loop variable:** See **index variable**.

**low-level language:** A programming language that is relatively close to the form the computer's processor can execute directly. One statement in a low-level language corresponds to a single machine-language instruction. Examples are 6502 machine language, 6502 assembly language, and 68000 machine and assembly languages. Compare **high-level language**.

**low-order byte:** The less significant half of a memory address or other two-byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the low-order byte of an address is usually stored first, and the **high-order byte** second. The opposite is true for Macintosh computers.

**low-power Schottky (LS):** A type of **transistor-transistor logic (TTL)** integrated circuit having lower power and higher speed than a conventional TTL integrated circuit; named for Walter Schottky (1886–1956), a semiconductor physicist.

**low-resolution graphics:** The display of graphics on a display screen as a 16-color array of blocks, 40 columns wide and 48 rows high. For example, on a Macintosh when the text window is in use, the visible low-resolution graphics display is 40 by 40 plotting points—that is, 40 by 40 **pixels**. See **high-resolution graphics**.

**LS:** See **low-power Schottky**.

**machine language:** The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 6502 microprocessor used in the Apple II family of computers) has its own form of machine language.

**mainframe computer:** A central processing unit or computer that is larger and more powerful than a minicomputer or a personal computer (microcomputer). Frequently called simply a *mainframe* for short. The Apple Access II program and MacTerminal make it possible to communicate with mainframe computers over telecommunications media.

**main logic board:** A large circuit board that holds RAM, ROM, the microprocessor, custom-integrated circuits, and other components that make the computer a computer.

**main memory:** The part of a computer's memory whose contents are directly accessible to the microprocessor; usually synonymous with **random-access memory (RAM)**. Programs are loaded into main memory, and that's where the computer keeps information while you're working. Sometimes simply called *memory*. See also **read-only memory**, **read-write memory**.

**MARK parity:** A bit of value 1 appended to a binary number for transmission. The receiving device checks for errors by looking for this value on each character. Compare **even parity**, **odd parity**.

**megabyte:** A unit of measurement equal to 1024 kilobytes, or 1,048,576 bytes; abbreviated Mb. See **kilobyte**.

**memory:** A hardware component of a computer system that can store information for later retrieval. See **main memory**, **random-access memory**, **read-only memory**, **read-write memory**.

**memory location:** A unit of main memory that is identified by an address and can hold a single item of information of a fixed size. In the Apple II family of computers, a memory location holds one byte, or eight bits, of information.

**memory-resident:** (1) Stored permanently in memory as firmware (ROM). (2) Held continually in memory even while not in use. DOS is a memory-resident program.

**menu:** A list of choices presented by a program, from which you can select an action.

**MHz:** Megahertz; one million hertz. See **hertz**.

**microcomputer:** A computer, such as any of the Apple II or Macintosh computers, whose processor is a **microprocessor**.

**microprocessor:** A computer **processor** contained in a single integrated circuit, such as the 6502 or 65C02 microprocessor used in the Apple II family of computers and the 68000 microprocessor used in the Macintosh family. The microprocessor is the **central processing unit (CPU)** of the microcomputer.

**microsecond:** One millionth of a second. Abbreviated  $\mu$ s.

**millisecond:** One thousandth of a second. Abbreviated ms.

**mode:** A state of a computer or system that determines its behavior. A manner of operating.

**modem:** Short for *MODulator/DEModulator*. A peripheral device that links your computer to other computers and information services using the telephone lines.

**modifier key:** A key (Apple, Caps Lock, Control, Option, Shift) that generates no keyboard events of its own, but changes the meaning of other keys or mouse actions. Also called a *control key*.

**modulate:** To modify or alter a signal so as to transmit information. For example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or *AM*) or the frequency (frequency modulation, or *FM*) of a carrier signal.

**monitor:** See **video monitor**.

**Monitor program:** A system program built into the firmware of some computers, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level. The Monitor program activates the disk drive when you turn on the computer.

**most significant bit:** The leftmost bit of a binary number. The most significant bit contributes the largest quantity to the value of the number. For example, in the binary number 10110 (decimal value 22), the leftmost bit has the decimal value 16 ( $2^4$ ). Compare **least significant bit**.

**mouse:** A small device you move around on a flat surface next to your computer. The mouse controls a pointer on the screen whose movements correspond to those of the mouse. You use the pointer to select menu items, to move data, and to draw with in graphics programs.

**mouse button:** The button on the top of the mouse. In general, pressing the mouse button initiates some action on whatever is under the pointer, and releasing the button confirms the action.

**nanosecond:** One billionth of a second. Abbreviated ns.

**nested loop:** A loop contained within the body of another loop and executed repeatedly during each pass through the outer loop. See **loop**.

**nested subroutine call:** A call to a subroutine from within the body of another subroutine.

**nibble:** A unit of data equal to half a byte, or four bits. A nibble can hold any value from 0 to 15.

**NOT:** A unary logical operator that produces a true result if its operand is false, and a false result if its operand is true. Compare **AND**, **OR**, **exclusive OR**.

**NTSC:** (1) Abbreviation for *National Television Standards Committee*. The committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

**object code:** See **object program**.

**object program:** The translated form of a program produced by a language translator such as a compiler or assembler. Also called *object code*. Compare **source program**.

**odd parity:** In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number; used as a means of error checking. Compare **even parity**, **MARK parity**.

**opcode:** See **operation code**.

**Open Apple:** A control key on the Apple II-family keyboards; on later keyboards, simply called the *Apple key*.

**operand:** A value to which an operator is applied. The value on which an operation code operates. Compare **argument**.

**operating system:** A program that organizes the actions of the parts of the computer and its peripheral devices. See **disk operating system**.

**operation code:** The part of a machine-language instruction that specifies the operation to be performed. Often called *opcode*.

**operator:** A symbol or sequence of characters, such as + or AND, specifying an operation to be performed on one or more values (the operands) to produce a result. See **arithmetic operator**, **relational operator**, **logical operator**, **unary operator**, **binary operator**.

**option:** (1) Something chosen or available as a choice; for instance, items in a menu. (2) An **argument** whose provision is optional.

**OR:** A logical operator that produces a true result if either or both of its operands are true, and a false result if both of its operands are false. Compare **exclusive OR**, **AND**, **NOT**.

**output:** Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

**output routine:** A machine-language routine that performs the sending of characters. The standard output routine sends characters to the screen. A different output routine might, for example, send them to a printer.

**overflow:** The condition that exists when an attempt is made to put more data into a given memory area than it can hold; for example, a computational result that exceeds the allowed range.

**override:** To modify or cancel an instruction by issuing another one.

**overrun:** A condition that occurs when the processor does not retrieve a received character from the receive data register of the Asynchronous Communications Interface Adapter (ACIA) before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

**page:** (1) A screenful of information on a video display. In the Apple II family of computers, a page consists of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256.

**page zero:** See **zero page**.

**parallel interface:** An interface in which several bits of information (typically 8 bits, or 1 byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

**parity:** Sameness of level or count, usually the count of 1 bits in each character, used for error checking in data transmission. See **even parity**, **MARK parity**, **odd parity**, **parity bit**.

**Pascal:** A high-level programming language with statements that resemble English phrases. Pascal was designed to teach programming as a systematic approach to problem solving. Named after the philosopher and mathematician Blaise Pascal.

**pass:** A single execution of a loop.

**PC board:** See **printed-circuit board**.

**peek:** To read information directly from a location in the computer's memory.

**peripheral:** (adj) At or outside the boundaries of the computer itself, either physically (as a peripheral device) or in a logical sense (as a peripheral card). (n) Short for *peripheral device*.

**peripheral bus:** The **bus** used for transmitting information between the computer and peripheral devices connected to the computer's expansion slots or ports.

**peripheral card:** A removable printed-circuit board that plugs into one of the computer's expansion slots. Peripheral cards allow the computer to use peripheral devices or to perform some subsidiary or peripheral function.

**peripheral device:** A piece of hardware—such as a video monitor, disk drive, printer, or modem—used in conjunction with a computer and under the computer's control. Peripheral devices are often (but not necessarily) physically separate from the computer and connected to it by wires, cables, or some other form of interface. They often require **peripheral cards**.

**peripheral slot:** See **expansion slot**.

**phase:** (1) A stage in a periodic process. A point in a cycle. For example, the 6502 microprocessor uses a clock cycle consisting of two phases called  $\Phi 0$  and  $\Phi 1$ . (2) The relationship between two periodic signals or processes.

**PILOT:** Acronym for *Programmed Inquiry, Learning, Or Teaching*. A high-level programming language designed for teachers and used to create computer-aided instruction (CAI) lessons that include color graphics, sound effects, lesson text, and answer checking. SuperPILOT is an enhanced version of the original Apple II PILOT programming language.

**pipelining:** A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All else being equal, processors with this feature run faster than those without it.

**pixel:** Short for *picture element*. A point on the graphics screen; the visual representation of a bit on the screen (white if the bit is 0, black if it's 1). Also, a location in video memory that maps to a point on the graphics screen when the viewing window includes that location.

**plotting vector:** A code representing a single step in drawing a shape on the high-resolution graphics screen. The plotting vector specifies whether to plot a point at the current screen position, and in what direction to move (up, down, left, or right) before processing the next vector. See **shape definition**, **shape table**.

**pointer:** An item of information consisting of the memory address of some other item. For example, Applesoft BASIC maintains internal pointers to the most recently stored variable, the most recently typed program line, and the most recently read data item, among other things. The 6502 uses one of its internal registers as a pointer to the top of the stack.

**point of call:** The point in a program from which a subroutine or function is called.

**poke:** To store information directly into a location in the computer's memory.

**pop:** To remove the top entry from a **stack**, moving the stack pointer to the entry below it. Synonymous with *pull*. Compare **push**.

**port:** In the Apple IIc, slots are called ports.

**power supply:** A circuit that draws electrical power from a power outlet and converts it to the kind of power the computer can use.

**power supply case:** The metal case inside most Apple II and Macintosh computers that houses the power supply. The Apple IIc uses an external power supply case.

**PR#:** An Applesoft BASIC command that sends output to a slot or a machine-language program. It specifies an output routine in the ROM on a peripheral card or in a machine-language routine in RAM by changing the address of the standard output routine used by the computer.

**precedence:** The order in which operators are applied in evaluating an expression. Precedence varies from language to language, but usually resembles the precedence rules of algebra.

**printed-circuit board:** A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly Fiberglas, to which integrated circuits and other electronic components are connected.

**procedure:** In the Pascal and Logo programming languages, a set of instructions that work as a unit; approximately equivalent to the term **subroutine** in BASIC.

**processor:** The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory. See **microprocessor**.

**ProDOS:** An Apple II operating system designed to support hard disk drives like the ProFile, as well as floppy disk storage devices. ProDOS stands for *Professional Disk Operating System*. Compare **Disk Operating System (DOS)**.

**ProDOS command:** Any one of the 28 commands recognized by ProDOS.

**program:** (n) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (v) To write a program.

**program line:** The basic unit of an Applesoft BASIC program, consisting of one or more statements separated by colons (:).

**programming language:** A set of symbols and associated rules or conventions for writing programs. BASIC, Logo, and Pascal are programming languages.

**prompt:** A message on the screen that tells you of some need for response or action. A prompt usually takes the form of a symbol, a message, a dialog box, or a menu of choices.

**prompt character:** A text character displayed on the screen, usually just to the left of a **cursor**, where your next action is expected. The prompt character often identifies the program or component of the system that's prompting you. For example, Applesoft BASIC uses a square bracket prompt character ([]); Integer BASIC, an angle bracket (>); and the system Monitor program, an asterisk (\*).

**prompt line:** A specific area on the display reserved for **prompts**.

**protocol:** A formal set of rules for sending and receiving data on a communication line.

**Protocol Converter:** A set of machine language routines used in the Apple II family for performing block device I/O. See **Smartport**.

**push:** To add an entry to the top of a **stack**, moving the stack pointer to point to it. Compare **pop**.

**queue:** A list in which entries are added at one end and removed at the other, causing entries to be removed in first-in, first-out (FIFO) order. Compare **stack**.

**QWERTY keyboard:** The standard layout of keys on a typewriter keyboard; its name is formed from the first six letters on the top row of letter keys. Compare **Dvorak keyboard**.

**radio-frequency (RF) modulator:** A device that makes your television set work as a monitor.

**RAM:** See **random-access memory**.

**random-access memory (RAM):** Memory in which information can be referred to in an arbitrary or random order. As an analogy, a book is a random-access storage device in that it can be opened and read at any point. RAM usually means the part of memory available for programs from a disk; the programs and other data are lost when the computer is turned off. A computer with 512K RAM has 512 kilobytes available to the user. (Technically, the read-only memory [ROM] is also *random access*, and what's called RAM should correctly be termed *read-write memory*.) Compare **read-only memory**, **read-write memory**.

**random-access text file:** A text file that is partitioned into an unlimited number of uniform-length compartments called *records*. When you open a random-access text file for the first time, you must specify its record length. No record is placed in the file until written to. Each record can be individually read from or written to—hence, *random-access*.

**raster:** The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive points on the individual lines of the raster.

**read:** To transfer information into the computer's memory from outside the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

**read-only memory (ROM):** Memory whose contents can be read, but not changed; used for storing **firmware**. Information is placed into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off. Compare **random-access memory**, **read-write memory**.

**read-write memory:** Memory whose contents can be both read and changed (or *written to*). The information contained in read-write memory is erased when the computer's power is turned off and is permanently lost unless it has been saved on a disk or other storage device. Compare **random-access memory**, **read-only memory**.

**real number:** In computer usage, a number that may include a fractional part; represented inside the computer in **floating-point** form. Because a real number is of infinite precision, this representation is usually approximate. Compare **integer**.

**receive data register:** A read-only register in the serial port ACIA (at \$C098 for port 1 and \$C0A8 for port 2) that stores the most recent character successfully received.

**register:** A location in a processor or other chip where an item of information is held and modified under program control.

**relational operator:** An operator, such as >, that operates on numeric values to produce a logical result. Compare **arithmetic operator**, **logical operator**.

**Request-To-Send:** An RS-232-C signal from a DTE to a DCE that serves to prepare the DCE for data transmission.

**reserved word:** A word or sequence of characters reserved by a programming language for some special use and therefore unavailable as a variable name in a program.

**resident:** See **memory-resident**, **disk-resident**.

**return address:** The point in a program to which control returns on completion of a subroutine or function.

**RF modulator:** See **radio-frequency modulator**.

**RGB monitor:** A type of color monitor that receives separate signals for each color (red, green, and blue). See **composite video**.

**ROM:** See **read-only memory**.

**routine:** A part of a program that accomplishes some task subordinate to the overall task of the program.

**row:** A horizontal arrangement of character cells or graphics **pixels** on the screen.

**RS-232 cable:** Any cable that is wired in accordance with the RS-232 standard, which is the common serial data communication interface standard.

**RTS:** See **Request-To-Send**.

**run:** (1) To execute a program. When a program *runs*, the computer performs the instructions.  
(2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

**save:** To store information by transferring the information from main memory to a disk. Work not saved disappears when you turn off the computer or when the power is interrupted.

**screen:** See **display screen**.

**scroll:** To move all the text on the screen upward or downward, and, in some cases, sideways. See **viewport**, **window**.

**serial interface:** An interface in which information is transmitted sequentially, a bit at a time, over a single wire or channel. Compare **parallel interface**.

**setup time:** The amount of time a signal must be valid in advance of some event. Compare **hold time**. See **valid signal**.

**silicon (Si):** A solid, crystalline chemical element from which integrated circuits are made. Silicon is a *semiconductor*; that is, it conducts electricity better than insulators, but not as well as metallic conductors. Silicon should not be confused with silica—that is, silicon dioxide, such as quartz, opal, or sand—or with silicone, any of a group of organic compounds containing silicon.

**simple variable:** A variable that is not an element of an array.

**68000:** The microprocessor used in the Macintosh and Macintosh Plus.

**6502:** The microprocessor used in the Apple II, in the Apple II Plus, and in early models of the Apple IIe.

**65C02:** The microprocessor used in the enhanced Apple IIe, the extended keyboard IIe, and the Apple IIC.

**slot:** A narrow socket inside the computer where you can install peripheral cards. Also called an **expansion slot**.

**Smartport:** A set of machine language routines used in the Apple II family for performing block device I/O. See **Protocol Converter**.

**soft switch:** Also called a *software switch*; a means of changing some feature of the computer from within a program. For example, **DIP switch** settings on ImageWriter printers can be overridden with soft switches. Specifically, a soft switch is a location in memory that produces some special effect whenever its contents are read or written.

**software:** A collective term for **programs**, the instructions that tell the computer what to do. They're usually stored on disks. Compare **hardware**, **firmware**.

**source code:** See **source program**.

**source program:** The form of a program given to a language translator, such as a compiler or assembler, for conversion into another form; sometimes called *source code*. Compare **object program**.

**space character:** A text character whose printed representation is a blank space, typed from the keyboard by pressing the Space bar.

**SPACE parity:** A bit value of 0 appended to a binary number for transmission. The receiving device can look for this value on each character as a means of error checking.

**stack:** A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order. Compare **queue**.

**standard instruction:** An instruction automatically present when no superseding instruction has been received.

**start bit:** A transition from a MARK signal to a SPACE signal for one bit-time, indicating that next string of bits represents a character.

**starting value:** The value assigned to the index variable on the first pass through a loop.

**start up:** To get the system running. Starting up is the process of first reading the operating system program from the disk, and then running an application program.

**startup disk:** A disk with all the necessary program files—such as the Finder and System files contained in the System folder in Macintosh—to set the computer into operation. In Apple II, sometimes called a *boot disk*.

**statement:** A unit of a program in a high-level language that specifies an action for the computer to perform. A statement typically corresponds to several instructions of machine language.

**status register:** A location in the ACIA (at \$C099 for port 1 and \$C0A9 for port 2) that stores the state of two RS-232-C signals and the state of the transmit and receive data registers, as well as the outcome of the most recent character transfer.

**step value:** The amount by which the index variable changes on each pass through a loop.

**stop bit:** A MARK signal following a data string (or the optional parity bit), indicating the end of a character.

**string:** An item of information consisting of a sequence of text characters.

**stroke:** A signal whose change is used to trigger some action.

**subroutine:** A part of a program that can be executed on request from another point in the program and that returns control, on completion, to the point of the request.

**synchronous:** A mode of data transmission in which a constant time interval exists between transmission of successive bits, characters, or events. Compare **asynchronous**.

**synchronous transmission:** A transmission process that uses a clocking signal to ensure an integral number of unit (time) intervals between any two characters. Compare **asynchronous transmission**.

**syntax:** (1) The rules governing the structure of statements or instructions in a programming language. (2) A representation of a command that specifies all the possible forms the command can take.

**system:** A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

**system configuration:** See **configuration**.

**system program:** A program that makes the resources and capabilities of the computer available for general purposes, such as an operating system or a language translator. Compare **application program**.

**system software:** The component of a computer system that supports application programs by managing system resources such as memory and I/O devices.

**tab:** An ASCII character that commands a device such as a printer to start printing at a preset location (called a *tab stop*). There are two such characters: horizontal tab (hex 09) and vertical tab (hex 0B). TAB works like the tabs on a typewriter.

**television set:** A display device capable of receiving broadcast video signals (such as commercial television broadcasts) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple II family of computers. Compare **video monitor**.

**text:** (1) Information presented in the form of readable characters. (2) The display of characters on a display screen. Compare **graphics**.

**text window:** An area on the video display screen within which text is displayed and scrolled.

**traces:** Electrical paths that connect the components on a circuit board.

**transistor-transistor logic (TTL):** (1) A family of integrated circuits having bipolar circuit logic; TTLs are used in computers and related devices. (2) A standard for interconnecting such circuits, which defines the voltages used to represent logical zeros and ones.

**transmit data register:** A location in the ACIA (at location \$C098 for port 1 and \$C0A8 for port 2) that holds the current character to be transmitted.

**troubleshoot:** To locate and correct the cause of a problem or malfunction, especially in hardware. Compare **debug**.

**TTL:** See **transistor-transistor logic**.

**turnkey disk:** See **startup disk**.

**unary operator:** An operator that applies to a single operand. For example, the minus sign (–) in a negative number such as –6 is a unary arithmetic operator. Compare **binary operator**.

**unconditional branch:** A branch that does not depend on the truth of any condition. Compare **conditional branch**.

**value:** An item of information that can be stored in a variable, such as a number or a string.

**variable:** (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location. Compare **constant**.

**vector:** (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

**video:** (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

**video monitor:** A display device that can receive video signals by direct connection only, and that cannot receive broadcast signals such as commercial television. Can be connected directly to the computer as a display device. Compare **television set**.

**viewport:** All or part of the display screen used by an application program to display a portion of the information (such as a document, picture, or worksheet) on which a program is working. Compare **window**.

**volume:** A general term referring to a storage device; a source of or a destination for information. A volume has a name and a volume directory with the same name. Its information is organized into files.

**warm start:** The process of transferring control back to the operating system in response to a failure in an application program. Compare **cold start**.

**window:** The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen. Compare **viewport**.

**word:** A group of bits that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

**write:** To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

**write-enable notch:** The square cutout on one edge of a 5.25-inch disk's jacket. If there is no write-enable notch, or if it is covered with a write-protect tab, the disk drive can read information from the disk, but cannot write on it.

**write protect:** To protect the information on a 5.25-inch disk by covering the write-enable notch with a write-protect tab, preventing the disk drive from writing any new information onto the disk. Compare **copy protect**.

**write-protect tab:** (1) A small adhesive sticker used to write protect a 5.25-inch disk by covering the write-enable notch. (2) The small plastic tab in the corner of a 3.5-inch disk jacket. You lock (write protect) the disk by sliding the tab toward the edge of the disk; you unlock the disk by sliding the tab back so that it covers the rectangular hole.

**X register:** One of the two index registers in the 6502 microprocessor.

**Y register:** One of the two index registers in the 6502 microprocessor.

**zero page:** The first page (256 bytes) of memory in the Apple II family of computers, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.



# Bibliography

*Addendum to the Design Guidelines.* Cupertino, Calif.: Apple Computer, Inc., 1984.

*Applesoft BASIC Programmer's Reference Manual*, Vols. 1 and 2. For the Apple II, IIe, and IIC. Cupertino, Calif.: Apple Computer, Inc., 1982. The version that applies to both the Apple IIe and the Apple IIC has Apple product number A2L0084 (Vol. 1) and A2L0085 (Vol. 2).

*Applesoft Tutorial.* Cupertino, Calif.: Apple Computer, Inc., 1982.

*Apple IIe Design Guidelines.* Cupertino, Calif.: Apple Computer, Inc., 1982.

*Apple II Monitors Peeled.* Cupertino, Calif.: Apple Computer, Inc., 1978. Currently not updated for Apple IIe and IIC, but a good introduction to Apple II-series input/output procedures; also useful for historical background.

Leventhal, Lance. *6502 Assembly Language Programming.* Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

*Synertek Hardware.* Santa Clara, Calif.: Synertek Incorporated, 1976. Does not contain instructions new to 65C02, but is the only currently available manufacturer's hardware manual for 6500-series microcomputers.

*Synertek Programming.* Santa Clara, Calif.: Synertek Incorporated, 1976. The only currently available manufacturer's programming manual for 6500-series microcomputers.

Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte*, Vol. 5, No. 11 (November 1980).

———. "A Simplified Theory of Video Graphics, Part II." *Byte*, Vol. 5, No. 12 (December 1980).

———. "More Colors for Your Apple." *Byte*, Vol. 4, No. 6 (June 1979).

- . "True Sixteen-Color Hi-Res." *Apple Orchard*, Vol. 5, No. 1 (January 1984).
- Wozniak, Steve. "System Description: The Apple II." *Byte*, Vol. 2, No. 5 (May 1977).
- . "SWEET16: The 6502 Dream Machine." *Byte*, Vol. 2, No. 10 (October 1977).



# Index

## Cast of Characters

- \* (asterisk) 59, 104
- @ (at sign) 113
- \ (backslash) 59, 63
- ^ (caret) 225
- : (colon) 216, 224
- \$ (dollar) 225
- . (period) 206
- ? (question mark) 59, 169, 179
- \_ (underscore) 179

## A

- accumulator 18, 64, 69, 84, 90, 115
- ACIA
  - block diagram 276
  - command register 280
  - control register 278–279
  - register locations for port 1 159
  - register locations for port 2 173
  - status register 130, 281
  - transmit/receive register 282
- acoustic coupler 177
- addresses
  - Applesoft BASIC interpreter 326
  - display 259–261
  - firmware 322–327
  - hardware 316–321, 353–356
  - I/O link 56–58
  - memory 20
  - mouse port 325
  - port 323–325
  - RAM 22, 351
  - ROM 22–23, 250, 351–352
  - serial port 323–324
  - video display 101
  - video firmware 324

- addressing modes 26, 226
  - 65C02 302, 304
- AltChar switch 102, 243, 360
- alternate character set 69, 70, 88, 92, 360
- AltZP switch 28, 242
- analog inputs 200
- any-key-down (AKD) 78, 243, 255
- Apple Integer BASIC 308, 330, 348, 356, 388
- Apple Language Card 351
- Apple Logo II 330
- Applesoft BASIC 169, 308, 329, 388
  - mouse and 195–196
- Applesoft BASIC interpreter 14, 23, 36, 51, 52, 59, 63, 204, 214, 250, 352
  - addresses 326
- Apple IIc
  - block diagram 235–236
  - schematic diagrams 291–296
- Apple II computers, interrupts 332
- Apple II series
  - differences 348–365
  - disk I/O 361
  - hardware 365
  - I/O 357
  - keyboard 357–359
  - machine identification 350
  - memory structure 351–356
  - video display 359–360
- A register 18, 43, 84, 113–115, 192, 213
- arithmetic, hexadecimal 215
- arrow keys 4
- ASCII character set 3, 70, 78, 80–81, 381–382, 391–395

- ASCII codes 58, 78, 80–81, 89, 164, 368
- ASCII input mode 209
- assembler 220
- assembly language, mouse and, 195–196
- assembly-language programs, debugging 221
- asterisk (\*) 59, 204
- asynchronous communications
  - interface adapter 130
- at sign (@) 113
- AUD 256, 365
- audio output jack 8, 256
- automatic repeat 3, 358
- Autostart ROM 356
- auxiliary memory 42–44, 74, 106, 160–161, 175, 269
  - screen holes 315
- auxiliary RAM 22, 184

## B

- back panel 9–10
- backslash (\) 59, 63
- backspace 63, 114
- BadBlock \$2D error 151
- BadCmd \$01 error 150
- BadCtl \$21 error 150
- BadCtlParm \$22 error 150
- BadPCnt \$04 error 150
- BadUnit \$11 error 150
- Bank2 switch 241
- bank selector switches 27–35
- bank-switched memory 24–35
- BASIC command 228
- baud rate
  - serial port 1 163
  - serial port 2 177–178

- bell 114
- bell character 115
- Bell routine 84
- Bell1 routine 84
- bits 384–386
- blanking intervals 257
- block device I/O firmware, entry points 23
- block-type devices 120
- BREAK signal 163
- BRK (\$00) instructions 212, 221, 334
  - handling 337–338
- buffers
  - display 38, 99
  - input 36, 38
  - serial I/O 343–345, 362
- BusErr \$06 error 150
- button interrupt mode 188
- bytes 384–386

## C

- CALL -151 command 223
- Canadian keyboard 375
- cancel line 63
- Caps Lock 5, 81, 358, 360
- caret (^) 225
- carriage return (CR) 63, 114, 164, 179
- carry bit 43
- cassette I/O 364
- C command 156, 170
- CH (cursor horizontal) 64
- character generator 14, 263
  - control signals 266
- character output switch (CSW) 57, 64, 71, 84, 101, 113, 115
- characters
  - at sign (@) 113
  - command 155, 205
  - control 4, 5, 60, 65–67, 70, 114, 165, 392
  - flashing 69, 70, 88–89
  - inverse 69, 70, 88–89
  - lowercase 395
  - normal 69, 70, 88–89
  - prompt 59
  - special 393
  - uppercase 394

- character sets 358–360
  - alternate 69, 70, 88, 92, 360
  - ASCII 3, 70, 78, 80–81, 381–382, 391–395
  - display 89
  - MouseText 91
  - primary 69, 70, 88, 359
  - screen 6
  - text 88–89
- ClampMouse routine 194
- ClearMouse routine 193
- CLEOLZ routine 112, 113
- clock rate 237
- clock signals 239–241
- CLOSE call 127, 141–142
- ClrEOL routine 112, 113
- ClrEOP routine 112, 113
- ClrScr routine 112, 113
- ClrTop routine 112, 113
- CmdNum 125
- CMOS 237
- code conversions 391
- cold start 51, 121–122
- colon (:) 216, 224
- colors
  - double high-resolution graphics 99
  - high-resolution 97, 268
  - low-resolution 94, 266
- column-address strobe (CAS) 252
- command character 155, 205
- command number 125, 127
- command register, ACIA 280
- connectors 9–10
  - disk drive 274
  - external power 234
  - hand controller 199–200, 287
  - keyboard 254
  - mouse 187, 284
  - serial port 154, 278
  - video expansion 270
  - video output 270
- CONTINUE BASIC command 228
- Control 5, 81, 255, 358
- Control-\ 67
- Control-[ 67
- Control-] 67
- Control-\_ 67
- Control-A 169, 171, 172, 179, 362
- Control-A I 181, 184

- Control-A Q 184
- Control-A T 180–182, 184
- Control-B 214, 228
- Control-C 67, 204, 214, 228, 341
- CONTROL call 127, 136–139
- control characters 4, 5, 60, 65–67, 70, 114, 165, 392
- Control-D 155, 169
- Control-E 213, 228
- Control-G 65, 66, 84
- Control-H 63, 65, 66
- Control-I 155, 158, 165, 362
- Control-J 65, 66
- Control-K 57, 66, 228
- Control-L 66
- control list 137, 138
- Control-M 65, 66, 118
- Control-N 66
- Control-O 66, 90
- Control-P 57, 90, 215, 228
- Control-Q 66
- Control-R 66, 172, 181, 184
- control register ACIA, 278–279
- Control-Reset 4, 49, 51, 52, 81, 121, 123, 162, 171, 176, 204, 218
- Control-S 66, 67, 341
- Control-T 171
- Control-U 63, 66, 82
- Control-V 66, 158, 172
- Control-W 66, 158
- Control-X 60, 63, 67, 82
- Control-Y 67, 218, 228
- Control-Z 67
- COut routine 63, 64, 90, 112, 113, 214
- COut1 routine 56, 60, 65, 67, 68, 70, 112, 114
- CP/M 328
- CPU 13
- CR *See* carriage return
- CROut routine 112, 114
- CROut1 routine 112, 114
- CSW 57, 64, 71, 84, 101, 113, 155
- CSWH 57
- CSWL 57
- C3COut1 routine 56, 61, 62, 65–67, 68, 70
- C3KeyIn routine 56, 58, 61, 62

- ul style="list-style-type: none;">
- cursor
  - blinking question mark 169, 179
  - blinking underscore 179
  - mouse 187
  - movement keys 4, 62
- CV (cursor vertical) 64
- D**
  - Data Carrier Detect 281
  - data format
    - serial port 1 163
    - serial port 2 177–178
  - data inputs 23
  - Data Set Ready (DSR) 281
  - Data Terminal Ready (DTR) 280
  - data transfer 42–43
  - debugging 212, 221
  - decimal 387–388
    - negative 388–389
  - Delete 4, 358
  - device control block (DCB) 130, 138
  - device information block (DIB) 131
  - DevSpec \$30–\$3F error 151
  - DHiRes switch 46, 102, 103
  - disassembler 220
  - disk controller unit 15, 247–248, 273, 365
  - disk drive 8–9
    - connector 274
    - connector signals 274
    - I/O 120–121, 273–274, 361
    - I/O port 120–121
  - disk-use light 3, 7
  - display addresses
    - mapping 259–261
    - transformation 260
  - display bits, high-resolution 96
  - display character sets 89
  - display formats
    - inverse 214
    - normal 214
  - display memory
    - addressing 258, 260
    - switches 44–46
  - display modes 104, 261–269, 360
    - switching 101–105
  - display pages 99–101
    - HRP1 38
    - HRP1X 38
    - HRP2 39, 269
    - HRP2X 39, 269
    - maps 105–111
    - TLP1 36
    - TLP1X 38
    - TLP2 38
    - TLP2X 38
  - display soft switches 50, 101–105
  - DisVBI switch 190
  - DisXY switch 189
  - DMA transfers 357
  - dollar sign (\$) 225
  - DOS 51, 57, 58, 155, 169, 204, 214, 312, 328, 332
  - Down Arrow 4
  - Dvorak keyboard 6–7, 358, 370
- E**
  - editing, GetLn 63
  - 80Col switch 101, 102, 243
  - 80-column display 38, 64, 68, 86, 91, 92, 100, 106, 358
    - addressing 263
    - dot patterns 269
    - map 108
    - switching to 5
  - 80/40 switch 3, 5
  - 80Store switch 39, 45, 101, 102, 106, 241
  - EnbXY switch 189
  - English keyboard 372
  - enhanced video firmware 249–250
  - EnlCRAM switch 242
  - entry points
    - firmware 23
    - memory expansion card 122
    - Monitor 322, 326–327
    - ports 71
    - UniDisk 3.5 I/O 122
  - EnVBI switch 190
  - environmental specifications 232
  - Escape 5, 60–62, 358
  - escape codes 60–62, 82
  - Escape Control-Q 215, 228
- EXAMINE command 213, 228
- expansion ROM space 74
- expansion slots 357
- Extended 80-Column Text Card 351, 354, 360
- EXTINT 276, 342
- F**
  - F command 156, 170
  - firmware 13
    - addresses 322–327
    - disk I/O 120
    - display mode 104
    - enhanced video 249–250
    - entry points 23
    - I/O 71–73
    - mouse 191–196
    - protocols 72–73
    - serial port 1 160
    - serial port 2 174
    - Smartport 124
  - firmware routines
    - Monitor 112–115
    - mouse 193–194
    - video 116–118
  - flag inputs 23
  - flags
    - inverse 69, 70
    - keyboard 78
  - flashing characters 69, 70, 88–89
  - forced cold-start reset 50, 52
  - FORMAT call 127, 135–136
  - Fortran 330
  - 40-column display 5, 50, 86, 91, 92, 94, 259, 358
    - addressing 263
    - map 107
    - memory 261
    - switching to 5
  - 48K memory 36–39
    - switches 39
  - French keyboard 373–374
  - full-duplex operation 182–184
  - function keys 4

## G

- game input 198–201
  - characteristics 199
- game paddles 198, 287
- GCR 247
- general logic unit (GLU) 15,  
245–246, 365
- German keyboard 376–377
- GetLn routine 36, 59–60, 78, 82,  
205
  - escape codes 60–62
- GetLn1 routine 82
- GetLnZ routine 82
- global storage 36
- GO command 219
- graphics display
  - buffer 38
  - double high-resolution 97–98,  
111, 269
  - high-resolution 38–39, 95–97,  
110, 267–268
  - low-resolution 94–95, 109,  
266–267
  - mixed-mode 98–99

## H

- half-duplex operation 180–181
- hand controller 198
  - circuits 288
  - connector 287
  - connector signals 199–200, 287
  - input and output 287–290,  
363–364
  - signals 289, 364
- handle 232
- hardware 365
  - addresses 316–321, 353–356
- headphone jack 8, 256
- hexadecimal 387–389
- hexadecimal arithmetic 215
- hexadecimal notation 225
- high-resolution graphics 38–39,  
95–97, 267–268
  - double 97–98, 111, 269
  - map 110
- HiRes switch 45, 103, 241, 243,  
354

- HLine routine 112, 114
- HomeMouse routine 194
- HOME routine 112, 114
- horizontal-count bits 259–260
- HRP1 38
- HRP1X 38
- HRP2 39, 269
- HRP2X 39, 269

## I

- I command 156, 170, 228
- identification bytes 73
- index registers 18
- INIT call 127, 139–140
- InitMouse routine 194
- IN#n command 57
- IN#2 command 169, 171, 172,  
176, 179, 180–183
- IN#4 command 196
- input buffer 36
- input/output unit (IOU) 15,  
243–244, 256, 365
- inputs
  - analog 200
  - data 23
  - flag 23
  - game 198–201
  - hand controller 287–290,  
363–364
  - keyboard 78–82
  - mouse 186–198, 282–286, 363
  - switch 200
- input subroutines 58–63
  - GetLn 59–60
  - KeyIn 58–59
  - Monitor 82
  - RdKey 58, 78, 341
- Integer BASIC 308, 330, 348,  
356, 388
- integrated circuits 13, 241–248
- Integrated Woz Machine (IWM) 15,  
247–248, 273, 365
- internal converter 234–235
  - specifications 234
- Interrupt Request 281

- interrupts 74, 331–347, 357
  - bypassing 345–347
  - external 342–343
  - EXTINT 276
  - handling 334–345
  - keyboard 341–342
  - mouse 50, 187–188, 190–191,  
339–340, 345–346, 354
  - serial 343–345
  - Smartport 130, 138
  - sources 338–339
  - VBIInt 187–188, 190, 243
  - vectors 333–334
  - XInt 190
  - YInt 190
- inverse characters 69, 70, 88–89
- INVERSE command 90, 214, 228
- inverse display 214
- inverse flag 69, 70
- I/O 357
  - cassette 364
  - disk 273–274, 361
  - disk drive 120–121
  - links 56–58
  - memory expansion card 123
  - port 71–74
  - serial 274–282, 361–362
  - Smartport 123–124
- IOError \$27 error 150
- IOUDis switch 45, 103, 189, 355
- IRQ 334
- ISO keyboard 371
- Italian keyboard 378–379

## J

- jacks
  - headphone 8, 256
  - output 8, 256
- joysticks 198, 199

## K

K command 156, 170  
keyboard 3-7, 254-255, 357-359  
    Canadian 375  
    characteristics 79  
    circuit diagram 254  
    Dvorak 6-7, 358, 370  
    encoder 78  
    English 372  
    features 3  
    flag 78  
    French 373-374  
    German 376-377  
    input buffer 36, 38  
    inputs 78-82  
    interrupts 340-342  
    ISO 371  
    Italian 378-379  
    layouts and codes 3, 6-7,  
        366-381  
    Sholes 6, 358, 367-368  
    signals 255  
    specifications 3  
    strobe 78, 339, 353  
    switch 3, 6-7  
    Western Spanish 380-381  
keyboard input switch (KSW) 57,  
    71, 101  
KeyIn routine 56-57, 58-59, 78  
keys  
    cursor movement 4, 62  
    modifier 5, 81  
    special function 4  
KSW 57, 71, 101  
KSWH 57  
KSWL 57

## L

languages 329-330  
last opened location 205  
L command 156, 170

Left Arrow 4, 63  
LF *See* line feed  
line feed 114, 164, 179  
LIST command 220-221, 225  
Logo II 330  
lowercase characters 395  
low-resolution graphics 94-95,  
    266-267  
map 109

## M

Machine Language Interface 125  
machine-language  
    programs 219-221  
main logic board 12-15  
main memory 42-44, 161, 175,  
    269  
    screen holes 313-314  
main RAM 22  
maps  
    display address 259-261  
    display page 105-111  
    48K memory 37  
    memory 20, 308-321  
    ROM 397, 398  
M command 156, 171  
memory  
    addressing 248-253  
    auxiliary 42-44, 74, 106,  
        160-161, 175, 269, 315  
    bank-switched 24-35  
    changing contents 208-210  
    comparing data 211-212  
    dump 206-208  
    examining 206  
    48K 36-39  
    main 42-44, 161, 175, 269,  
        313-314  
    maps 20, 308-321  
    moving data 210-211  
    RAM 13  
    range 207  
    ROM 13

memory addresses 20  
    display 258, 260  
    hardware page 316-321  
    port I/O 72-73  
    port screen hole 74  
    text window 68-69  
memory bus organization 249  
memory expansion 14  
    mouse and 191, 195  
memory expansion card 120, 250,  
    291  
    entry points 122  
    I/O 123  
    startup routine 122  
memory management unit  
    (MMU) 15, 241-242, 365  
memory pages  
    \$00 (zero page) 20, 24, 25,  
        308-311  
    \$01 (stack) 20, 24, 25  
    \$02 (input buffer) 36  
    \$03 (global storage, vectors) 36,  
        312  
    \$04-\$07 (TLP1) 36-38  
    \$08 (communication port  
        buffers) 38  
    \$08-\$0B (TLP2) 38  
    \$20-\$3F (HRP1) 38  
    \$40-\$5F (HRP2) 39  
    \$D0-\$FF 26  
microprocessor 13, 18-19  
    *See also* 65C02 microprocessor  
Mini-Assembler  
    address formats 226  
    instruction formats 226  
    starting 223-224  
    using 224-226  
mixed-mode displays 98-99  
MIXED switch 102, 243, 354  
MLI 125  
modem 177-178  
modem port, commands 170-172  
modifier keys 5, 81

- Monitor 5, 23, 36, 57, 59, 63, 161, 176, 258, 312, 356
  - entry points 322, 326–327
  - firmware routines 112–115
  - game support 201
  - input routines 82
  - interrupts 74
  - memory location 204
  - output 270
  - speaker routines 84
  - vectors 326–327

- Monitor commands
  - advanced 216–218
  - debugging 221
  - machine-language
    - program 219–221
  - memory 205–212
  - register 212–213
  - repeating 217–218
  - summary 227–229
  - syntax 205

- mouse
  - as hand controller 198
  - assembly-language
    - support 195–196
  - BASIC support 195–196
  - button signals 286
  - circuits 285
  - connector 284
  - connector signals 284
  - defaults 50
  - firmware entry points 23
  - firmware routines 193–194
  - inputs 186–198, 282–286, 363
  - interrupts 50, 339–340, 345–346, 354
  - I/O firmware support 191–196
  - operating modes 187–188
  - Pascal support 195
  - port characteristics 186
  - screen holes 196–197
  - soft switches 189–191
  - waveforms 283
- mouse port
  - addresses 325
  - I/O firmware protocol 195
  - screen holes 197
- MouseText 69, 70, 88, 90, 360
  - characters 91

- MouX1 switch 190
- MouY1 switch 190
- MoveAux routine 42–43
- MOVE command 210–211, 216–217, 227
- movement/button interrupt
  - mode 188
- movement interrupt
  - mode 187–188
- multiplexing 251
  - RAM address 252

## N

- N command 228
- nD command 156, 170
- negative decimal 388–389
- next changeable location 205, 208–209
- nibbles 94, 384, 386
- nnB command 156, 170
- nnn command 156, 170
- nnnN command 157, 171, 362
- NoDrive \$28 error 151
- NonFatal \$50–\$7F error 151
- normal characters 69, 70, 88–89
- NORMAL command 90, 214, 228
- normal display 214
- NoWrite \$2B error 150
- nP command 157, 171
- NTSC 86, 96, 97, 257, 270
- #6 command 155
- #7 command 169
- #8 command 169

## O

- O Control-K 215
- OffLine \$2F error 151
- Open Apple 4, 52, 81, 200, 221, 222, 358
- Open Apple-Control-Reset 52, 121, 162, 176, 361
- OPEN call 127, 140–141
- operating systems 214, 328–329
- output jack 256

- outputs
  - hand controller 363–364
  - speaker 82–84
  - strobe 24
  - video display 116–118
  - video signal 270–273
- output subroutines 64–70
  - COut 64
  - COut1 65
  - C3Out1 65–67
- overflow bit 44

## P

- paddles 198, 287
- Page 1 20, 36–38, 50, 269
  - high-resolution 95
  - text 100, 106
- Page 1X 38, 269
  - text 100
- Page 2 20, 39
  - high-resolution 95
  - text 100
- Page2 switch 45, 102, 241, 243, 354
- Page 2X 38, 39
- pages, display *See* display pages
- pages, memory *See* memory pages
- PAL 257
- parity bit 163
- Pascal 56, 68, 72, 124, 169, 330, 332
  - mouse and 195
  - video control functions 117–118
- Pascal Operating System 329
- PdI0 287
- PdI1 287
- PEEK 329
- period (.) 206
- peripheral-card memory
  - spaces 352–353
- peripheral identification numbers (PIN) 389–390
- PInit routine 116

- ul style="list-style-type: none;">
- pinouts
  - GLU 246
  - IOU 243
  - IWM 247
  - MMU 242
  - RAM 251
  - ROM 249, 250
  - 6551 277
  - TMG 245
  - video expansion connector 272
- PLOT routine 112, 114
- POKE 329
- ports
  - characteristics 71
  - disk I/O 120–121
  - entry points 71
  - I/O 71–74
  - mouse 186, 325
  - printer 155–157
  - ROM space 73
  - screen hole RAM space 74
  - serial 275
  - serial port 1 22, 154–165
  - serial port 2 22, 38, 167–184
  - video output 86
- PosMouse routine 193
- power converter 234–235
  - specifications 234
- power light 3, 7
- power-on reset 50
- power supply 11–12
  - specifications 233, 383
- PR#n command 57
- PR#0 command 198
- PR#1 command 155, 159, 162, 181
- PR#2 command 169, 171, 172, 179, 181, 183, 184
- PR#3 command 90, 196
- PR#4 command 196, 198
- PR#7 command 328
- PRBI2 routine 112, 114
- PrByte routine 112, 115
- PRead routine 116, 201
- P register 18, 213
- PrErr routine 112, 115
- PrHex routine 112, 115
- primary character set 69, 70, 88, 359
- PRINTER command 155
- printer port, commands 155–157
- PrntAX routine 113, 115
- processor status register 18, 213
- ProDOS 51, 57, 124, 155, 169, 204, 214, 312, 328, 332
- Machine Language Interface 125
- program counter (PC) 18
- prompt characters 59
- Protocol Converter 124
- PStatus routine 118
- PTrig switch 190
- PWrite routine 117
- Q**
- Q command 171
  - question mark (?) 59
    - blinking 169, 179
- R**
- RAM 13
    - addressing 251–253
    - auxiliary 22, 184
    - main 22
    - memory expansion card 123
    - peripheral-card 353
    - port screen hole 74
    - refreshing 251–252
    - Smartport 126
    - timing 252–253
  - RAM addresses 22, 351
    - multiplexing 252
  - RAMRd switch 39, 44, 242
  - RAMWrt switch 39, 44, 242
  - R command 157, 171
  - Rd63 switch 190
  - Rd80Col switch 102
  - Rd80Store switch 45, 102
  - RdAltChar switch 102
  - RdAltZP switch 28
  - RdBnk2 switch 28
  - RdBtn0 switch 190
  - RdChar routine 82
  - RdDHiRes switch 46, 103
  - RdHiRes switch 45, 103
  - RdIOUDis switch 46, 103, 189
  - RdKey routine 58, 78, 341
  - RdLCRAM switch 28
  - RdMIXED switch 102
  - RdPage2 switch 45, 102
  - RdRAMRd switch 39
  - RdRAMWrt switch 39
  - RdTEXT switch 102
  - RdVBImsk switch 190
  - RdX0Edge switch 189
  - RdXYMsk switch 189
  - RdY0Edge switch 190
  - READ BLOCK call 126, 132–133
  - READ call 127
  - ReadMouse routine 188, 192, 193
  - registers 18–19
    - changing 213
    - command 280
    - control 278–279
    - examining 213
    - shift 263, 268
    - Smartport 125
    - status 18, 130, 213, 281
    - transmit/receive 282
  - REMIN command 169
  - REMOUT command 169
  - Request to Send (RTS) 280
  - Reset 3, 4, 81, 358
  - reset routine 49–50
  - reset vectors 50–53
  - Return 4, 60, 169, 204, 206, 208, 209, 213, 215, 224, 227
  - return from subroutine (RTS) 57, 219
  - retype 63
  - RGB 269
  - Right Arrow 4, 63, 82
  - ROM 13
    - addresses 22–23, 351–352
    - addressing 249–250
    - Autostart 356
    - character generator 14, 266, 267
    - Monitor 356
    - peripheral-card 352

- ROMEN1 250
- ROMEN2 250
- ROM map
  - auxiliary side 398
  - main side 397
- ROM space
  - expansion 74
  - port 73
- row-address strobe (RAS) 252
- RstVBI switch 190
- RstXInt switch 189, 242
- RstXY switch 189
- RstYInt switch 190, 242
- RTS
  - Request to Send 280
  - return from subroutine 57, 219

## S

- S command 157, 171
- screen holes 38, 312–315
  - auxiliary memory 315
  - main memory 313–314
  - mouse 196–197
  - serial port 1 160–161
  - serial port 2 174–175
- SCRN routine 113, 115
- serial buffering 343–344
- serial cards 361–362
- serial I/O 274–282, 361–362
  - buffers 362
- serial port 361–362
  - circuits 275
  - connectors 278
  - connector signals 278
- serial port 1 22, 154–165
  - addresses 323
  - at startup 159
  - carriage return and line feed 164
  - characteristics 154, 161–165
  - data format and baud rate 163
  - displaying output 165
  - hardware page locations 159
  - I/O firmware support 160
  - screen holes 160–161
  - sending special characters 165
  - using 155–158

- serial port 2 22, 38, 167–184
  - addresses 324
  - at startup 173
  - carriage return and line feed 179
  - characteristics 168–169, 176–184
  - commands 170–172
  - data format and baud rate 177–178
  - hardware page locations 173–174
  - I/O firmware support 174
  - routing input and output 179–184
  - screen holes 174–175
  - using 169–173
- ServeMouse routine 187, 188, 193, 339
- SetCol routine 113, 115
- SetMouse routine 192, 193, 339
- SetPWRC routine 53
- Shift 5, 81, 255, 358
- shift register 263, 268
- Sholes keyboard 6, 358, 367–368
- signals
  - BREAK 163
  - character generator 266
  - clock 239–241
  - disk drive connector 274
  - GLU 246
  - hand controller 199–200, 289, 364
  - hand controller connector 287
  - IOU 243–244
  - IWM 247
  - keyboard 255
  - MMU 242
  - mouse button 286
  - mouse connector 187, 284
  - RAM timing 253
  - ROM 250
  - serial port connector 278
  - 65C02 305
  - 6551 277
  - synchronization 257
  - timing 264–265
  - TMG 245
  - video expansion
    - connector 272–273
  - video output 270–273

- 65C02 microprocessor 13, 75, 237–241, 297–307
  - addressing modes 26, 302, 304
  - block diagram 237–238, 299
  - characteristics 300–301
  - data sheets 298–307
  - enhancements 301
  - instruction mnemonics 302
  - instruction set 306
  - interrupt handling 333
  - operational codes 306–307
  - pin configuration 299
  - pin function 300
  - programming model 303
  - ratings 300
  - registers 18–19
  - signal description 305
  - specifications 239
  - timing 239–241, 302
  - versus 6502 297–298
- SLOT3ROM switch 354
- SLOTXROM switch 354
- Smartport
  - calls 125–148
  - commands and parameters 149
  - error codes 150–151
  - I/O interface 123–124
  - locating 124
  - soft switches 20, 24
  - bank selector 27
  - display 50, 101–105
  - mouse 189–191
- Solid Apple 4, 52, 81, 200, 221, 222, 358
- Solid Apple-Control-X 341
- speaker 359
  - circuit diagram 256
  - outputs 82–84
  - volume control 8, 256
- special characters 393
- special function keys 4
- specifications
  - environmental 232
  - power supply 233, 383
  - 65C02 239
- S register 18, 213
- stack pointer 18, 213
- startup 121–123
- STATUS call 126, 128–132

- status register, ACIA 130, 281
- STEP command 221–223
- stop-list function 67
- STORE command 216, 227
- strobes
  - column-address 252
  - keyboard 78, 255, 339, 353
  - outputs 24
  - row-address 252
- Super Serial Card 179, 361, 362
- Sw0 287
- Sw1 287
- switch inputs 200
- synchronization signals 257

**T**

- Tab 4, 358
- T command 171, 180–183
- terminal mode 179, 184
- text, character sets 88–89
- text display 263
  - characteristics 93
  - 40-column 5, 50, 86, 91, 92, 94, 107, 259, 261, 358
  - 80-column 5, 38, 64, 68, 86, 91, 92, 100, 106, 108, 269, 358
  - switching 93
- text Page 1 100, 106
- text Page 1X 100
- text Page 2 100
- TEXT switch 102, 243, 354
- text window 68–69
- timing generator (TMG) 15, 245, 365
- TLP1 36
- TLP1X 38
- TLP2 38
- TLP2X 38
- toggle switches 24
- TRACE command 221–223
- transmit/receive register, ACIA 282
- transparent mode 187

**U**

- underscore ( \_ ), blinking 179
- UniDisk 3.5 120, 124, 130, 223
  - control list 138
  - I/O entry points 122
- Up Arrow 4
- uppercase characters 394
- USER command 218, 228

**V**

- validity-check byte 51, 53
- VBIInt switch 243
- vectors 36, 56
  - interrupt 333–334
  - Monitor 326–327
  - reset 50–53
- VERIFY command 211–212, 217, 227
- vertical blanking 338, 354, 360
  - active modes 188
- vertical-count bits 260
- VID 270, 365
- video counters 257–258
- video display 257–273, 359–360
  - addresses 101
  - circuits 262
  - I/O firmware support 116–118
  - modes 261–269
  - specifications 87
- video expansion
  - connector signals 272–273
  - output 271–273
- video output firmware
  - addresses 324
  - entry points 23
- video output port,
  - characteristics 86
- video output signals 257, 270–273
- VLine routine 113, 115
- voltage converter 11–12
- volume control 8, 256
- VTabZ routine 113

**W**

- warm start 50, 51, 123
- Western Spanish
  - keyboard 380–381
- WNDW 257
- words 385
- WRITE BLOCK call 127, 134–135
- WRITE call 127, 143–144

**X**

- X0 243, 284
- X0Edge switch 189
- X1 284
- X command 157, 171
- XFer routine 42–44, 312
- XInt 190, 354
- XON/XOFF protocol 171
- X register 18, 43, 73, 84, 114, 115, 118, 128, 192, 201, 213

**Y**

- Y0 243, 284
- Y0Edge switch 190
- Y1 284
- YInt 190, 354
- Y register 18, 43, 113, 114, 115, 128, 192, 201, 213

**Z**

- Z command 157, 165, 171

## THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh™ Plus and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter® Plus. POSTSCRIPT™, the LaserWriter's page-description language, was developed by Adobe Systems Incorporated.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Program listings are set in Apple Courier, a monospaced font.

# The Apple Technical Library

## The Official Publications from Apple® Computer, Inc.

The Apple Technical Library offers programmers, developers, and enthusiasts the most complete technical information available on Apple computers, peripherals, and software. The Library consists of technical manuals for the Apple II family of computers, the Macintosh family of computers, key peripherals, and programming environments.

Apple Technical Library titles on the Apple II family include technical references to the Apple IIe, Apple IIc, and Apple IIgs computers, with detailed descriptions of the hardware, firmware, ProDOS operating system, and the built-in programming tools that programmers and developers can draw upon. In addition to a technical introduction and programmer's guide to the Apple IIgs, there are tutorials and references for Applesoft BASIC and Instant Pascal programmers.

The Inside Macintosh Library provides complete technical references to the Macintosh 512, Macintosh 512 enhanced, Macintosh Plus, Macintosh SE, and Macintosh II computers. Individual volumes provide technical introductions and programmer's guides to the Macintosh as well as detailed information on hardware, firmware, system software, and programming tools. The Inside Macintosh Library offers the most detailed and complete source of information available for the Macintosh family of computers.

In addition, titles in the Apple Technical Library offer references to the wide range of important printers, communications standards, and programming environments such as the Standard Apple Numerics Environment (SANE) to help programmers and experienced users get the most out of their computer systems.



# Apple IIc Technical Reference Manual

The Official Publication from Apple Computer, Inc.

Apple's definitive guide to all versions of the Apple® IIc personal computer. Written and produced by the people at Apple Computer, this manual provides a comprehensive, single-source reference for programmers and hardware designers.

The *Apple IIc Technical Reference Manual* describes all aspects of the IIc—including its physical characteristics, the hardware/firmware locations that control memory and I/O, and the electrical and electronic implementation of machine features and capabilities. Summary tables provide quick reference to all of the I/O firmware entry points, including those used by the system's Smartport.

The manual describes:

- Memory organization and control.
- Apple IIc I/O interface, including keyboard and speaker, video display modes (including graphics modes), internal and external disk drives, the Apple IIc Memory Expansion Card, serial ports, and the mouse and game port.
- Use of the system Monitor routines in the IIc firmware to disassemble and debug machine-language programs.
- The hardware, including pinouts of custom ICs and internal/external connectors, plus schematic diagrams.

Appendixes contain quick-reference tables, describe interrupt handling, and include product comparisons of all members of the Apple II family of computers.

This edition of the *Apple IIc Technical Reference Manual* also includes firmware listings for the new Apple IIc Memory Expansion Card. Information on obtaining firmware listings for earlier versions of the IIc, as well as supplementary technical information, is provided.

**Apple Computer, Inc.**  
20525 Mariani Avenue  
Cupertino, California 95014  
(408) 996-1010  
TLX 171-576

030-1238-B  
Printed in U.S.A.

Addison-Wesley Publishing Company, Inc.

ISBN 0-201-17752-8